UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: Advanced Engineering Mathematics **Subject Code-MCSE101**
Year: 1st Year **Semester: first**

Module Number	Topics	Number of Lectures (27)
1	**NumericalAnalysis:**	**4L**
	IntroductiontoInterpolationformulae: Stirling Bessel's Spline	2
	Solutionsofsystemoflinearandnon-linear simultaneousequations: SORalgorithm Newton'smethod	2
2	**Stochasticprocess:**	**8L**
	Probability:review,randomvariables,randomprocesses,Randomwalk,brownianmotion,markovprocess,	4
	queues:(M/M/1):(/FIFO),(M/M/1):(N/FIFO)	4
3	**Advancedlinearalgebra**:	**9L**
	Vector spaces, linear transformations, eigenvalues, Eigenvectors, some applications of eigenvalue problems	4
	symmetric,skew-symmetricAndorthogonalmatrices,similarityofmatrices	2
	basisofEigenvectors,diagonalisation	3
4	**AdvancedGraphTheory:**	**6L**
	Connectivity,Matching,HamiltonianCycles,ColoringProblems	4
	Algorithmsforsearchinganelementinadata structure(DFS,BFS)	2

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: **Advanced Operating System** Subject Code: **MCSE 102**

Year: **1st Year** Semester: **First**

Module Number	Topics	Number of Lectures
1	Operating System Introduction, Structures - Simple Batch, Multi programmed, time-shared, Personal Computer, Parallel, Distributed Systems, Real-Time Systems, System components, Operating-System services, System Calls, Virtual Machines, System Design and Implementation.	4L
	Process and CPU Scheduling - Process concepts and scheduling, Operation on processes, Cooperating Processes, Threads, and Interposes Communication Scheduling Criteria, Scheduling Algorithm, Multiple -Processor Scheduling, Real-Time Scheduling	5L
2	Memory Management and Virtual Memory - Logical versus Physical Address Space, Swapping, Contiguous Allocation, Paging, Segmentation, Segmentation with Paging. Demand Paging, Performance of Demanding Paging, Page Replacement, Page Replacement Algorithm, Allocation of Frames, Thrashing	6L
	File System Interface and Implementation - Access methods, Directory Structure, Protection, File System Structure, Allocation methods, Free-space Management, Directory Management, Directory Implementation, Efficiency and Performance	6L
3	Deadlocks - System Model, Dead locks Characterization, Methods for Handling Dead locks Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, and Recovery from Deadlock	4L
	Process Management and Synchronization - The Critical Section Problem, Synchronization Hardware, Semaphores, and Classical Problems of Synchronization, Critical Regions, Monitors	5L

4	Operating System Security Issues- Introduction to the topic of Security in Operating Systems, Principles of Information Security, Access Control Fundamentals, Generalized Security Architectures	5L
5	Introduction to Distributed systems: Goals of distributed system, hardware and software Concepts, design issues	2L
	Elementary introduction to the terminologies within Modern Oss: Parallel, Distributed, Embedded & Real Time, Mobile, Cloud and Other Operating System Models	3L
Total Number Of Lectures = 40		

Faculty In-Charge **HOD, CSE Dept.**

Assignments:

Module-1:

1. Explain Thread with real life examples.
2. Discuss real time scheduling.

Module-2:

1. What are the differences between paging and segmentation?
2. When does a page fault occur? Explain various page replacement strategies.

Module-3:

1. Why does starvation occur? What is the solution of starvation?
2. Discuss all the deadlock avoidance techniques?
3. State the roles of binary and count semaphore.

Module-4:

1. Discuss the security issues of OS?

Module-5:

1. Define real time operating system and its applications in real life?
2. Differentiate hard real time system from soft real time system.
3. Define distributed OS? How DOS differs from general purpose OS?
4. Write down the short notes on the following topics
   A. POSIX
   B. Event Driven Scheduling
   C. Sporadic Task
   D. VX-Works

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: Advance Computer Architecture **Subject Code: MCSE103**
Year: 1st Year **Semester: 1st**

Module Number	Topics	Number of Lectures
1	**The evolution of modern Computer systems & Introduction to high performance Computing**	**12**
	1. DEC PDP-11, IBM 360/370 family, CDC Cyber 6600, Intel X86 architecture	2
	2. Performance measurement parameters – MIPS, MFLOPS, SPEC ratings, CPI etc	2
	3. Overview, Flynn's classifications – SISD, SIMD, MISD, MIMD	1
	4. Examples from Vector & Array Processors	1
	5. Performance comparison of algorithms for Scalar, Vector and Array Processor	2
	6. Fundamentals of UMA, NUMA, NORMA architectures Performance measurement for parallel architectures – Flynn's measure, Feng's measure, Handler's measure	2
	7. Amadahl's law of limitation for parallel processing, Gustafson's law	2
2	**Pipelined processor design**	**12**
	1. Pipeline performance measurement parameters – speedup factor, efficiency, throughput of a linear pipeline, comparing performance of a N stage pipeline with a N processor architecture	2
	2. Pipeline design principles – Uniform sub computations, Identical computations, Independent computations	2
	3. Examples from design of Arithmetic pipelines – Floating point Adders, Multipliers, Dividers etc.	2
	4. Classifications of Uni function, Multifunction & Dynamic pipelines	2
	5. Scheduling in a pipelines with feedback	2
	6. Pipeline hazards and their solutions	2
3	**Various architectures**	**10**
	1. RISC architecture, characteristics of RISC instruction set & RISC pipeline, its comparisons with CISC, necessity of using optimizing compilers with RISC architecture	2
	2. Examples from POWER PC and SPARC architectures	2
	3. Super pipelining (MIPS architecture),	2

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: Advance Computer Architecture **Subject Code: MCSE103**
Year: 1st Year **Semester: 1st**

	Superscalar architecture 4. Diversified pipelines and out of order execution, VLIW architecture 5. Hardware multithreading (Coarse grained, fine grained & simultaneous multithreading)	 2 2
4	**Memory hierarchy**	**6**
	1. Techniques for improving Cache memory performance parameters,(reduce cache miss rate, reduce hit time, reduce miss penalty) 2. Main memory performance enhancement – interleaved memory, improvement of memory bandwidth 3. Use of TLB for performance enhancement	3 2 1
Total Lecture Hours – 34 l.h.		

Faculty In-Charge HOD, CSE Dept.

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: Advance Computer Architecture **Subject Code: MCSE103**
Year: 1st Year **Semester: 1st**

Assignments:-

Unit 1:-

1. Explain the difference between hardwired and control and micro programmed control.
2. Explain the importance of different addressing modes in computer architecture with suitable example.
3. What is an instruction format? Explain different types of instruction formats in detail.
4. Explain Flynn's classification of computers.
5. Formulate a hardware procedure for detecting an overflow by computing the sign of the sum with the signs of the augends and addend. The numbers are in signed 2"s complement representation.
6. Explain vector processing. What is the difference between vector & array processing?
7. What is the difference between serial and parallel transfer? Explain with the required example.
8. A digital computer has a common bus system for 16 register of 32 bits each. The bus is constructed with multiplexers.

(i) How many selection inputs are there in each multiplexer?
(ii) What sizes of multiplexers are needed?
(iii) How many multiplexers are there in the bus?

9. What is the difference between a direct and indirect address instruction? How many references to memory are needed for each type of instruction to bring an operand into a processor register?
10. A computer has 16 register, an ALU with 32 operations and a shifter with eight operations all Connected to a common bus system.

(i) Formulate a control word for a micro operation.
(ii) Specify the number of pits in each field on the control word and give a general encoding scheme.

11. List down the problems with MIPS system.
12. Discuss the Amadahl's law with proper example. Discuss fraction and speed up with real time case study.
13. Discuss NUMA and UMA with their properties.
14. Design a (very) simple CPU for an instruction set that contains only the following four instructions: lw (load word), sw (store word), add, and jump (unconditional branch). Assume that the instruction formats are similar to the MIPS architecture. If you assume a different format, state the instruction formats. Show all the components, all the links, and all the control signals in the datapath. You must show only the minimal hardware required to implement these four instructions. For each instruction show the steps involved and the values of the control signals for a single cycle implementation.

Unit 2:-

1. Draw a space time diagram for a six-segment pipeline showing the time it takes to process eight Tasks.
2. A no pipeline system takes 50 ns to process a task. The same task can be processed in 6 segment pipeline with a clock cycle of 10 ns. Determine the speedup ratio of pipeline for 100 tasks. What is maximum speedup ratio?
3. Explain hazards to the instruction pipeline with their solution.

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: Advance Computer Architecture **Subject Code: MCSE103**
Year: 1st Year **Semester: 1st**

4. What do you mean by instruction cycle and interrupt cycle? Draw the flowchart for instruction Cycle.
5. Determine the number of clock cycles that it takes to process 200 task in a six segment pipeline.
6. Consider the following MIPS assembly code:

   LD R1, 45(R2)
   ADD R7, R1, R5
   SUB R8, R1, R6
   OR R9, R5, R1
   BNEZ R7, target
   ADD R10, R8, R5
   XOR R2, R3, R4

a) Identify each type of data dependency; list the two instructions involved; identify which instruction is dependent; and, if there is one, name the storage location involved.

b) Use MIPS five-stage pipeline (fetch, decode, register, execute, write-back) and assume a register file that writes in the first half of the clock cycle and reads in the second half cycle. Which of the dependencies that you found in part (a) become hazards and which do not? Why?

Unit 3:-

1. a)Name two RISC and two CISC processors. What are the main characteristics of RISC processors?

   b) Define (i) superscalar and (ii) super-pipeline concepts. Derive the equation for ideal speedup for a superscalar super-pipelined processor compared to a sequential processor. Assume N instructions, k-stage scalar base pipeline, superscalar degree of m, and super pipeline degree of n.

2. Draw a diagram showing how the instruction fetch and execution units of a superscalar processor are connected. Show the widths of the datapath (in words - not bits; your diagram should be relevant to a 32-bit or 64-bit processor). Which factor primarily determines performance: the instruction issue width (number of instructions issued per cycle) or the number of functional units?
3. List the capabilities of the instruction fetch/despatch unit needed to make an effective superscalar processor.
4. Why does a VLIW machine need a good optimizing compiler?
5. Where can you find a small dataflow machine in every high performance processor?
6. How the processor performance can be improved other than the enhancement in VLSI technology?
7. Why does a VLIW machine restrict the op-codes which may be placed in any 'slot' of its instructions?

Unit 4:-

1. Give an example of a situation where caches in a shared memory machine need to be kept coherent.
2. List the states that a cache line may be in for the most commonly implemented cache coherence protocol. Provide a short description of each state.

3. What is the advantage of having both **Exclusive** and **Shared** states for cache lines in a shared memory machine?
4. What is the name of the programming model most commonly used for distributed memory machines?
5. A three-level memory system having cache access time of 5 nsec and disk access time of 40 nsec, has a cache hit ratio of 0.96 and main memory hit ratio of 0.9. What should be the main memory access time to achieve an overall access time of 16 nsec ?
6. According to the following information, determine size of the subfields (in bits) in the address for Direct Mapping and Set Associative Mapping cache schemes :
   We have 256 MB main memory and 1 MB cache memory
   The address space of the processor is 256 MB
   The block size is 128 bytes
   There are 8 blocks in a cache set.
7. Suppose physical addresses are 32 bits wide. Suppose there is a cache containing 256K words of data (not including tag bits), and each cache block contains 4 words. For each of the following cache configurations,
   a. direct mapped
   b. 2-way set associative
   c. 4-way set associative
   d. fully associative
   specify how the 32-bit address would be partitioned. For example, for a direct mapped cache, you would need to specify which bits are used to select the cache entry and which bits are used to compare against the tag stored in the cache entry.

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: Adv. Algorithm **Subject Code-MCSE104**
Year: 3rd Year **Semester: Fifth**

Module Number	Topics	Number of Lectures
1	**Complexity Analysis:**	**3L**
	1. Time and space complexity. Asymptotic notations. Recurrence for divide and conquer And its solution, the substitution method and recursion-tree method for solving recurrences. The master method with proof	1
	2. Solve recursive function with different methods	2
2	**Advanced data structure: :**	**3L**
	1. ADT and data structure, linear vs non-linear data Structure. Tree: tree as an ADT, definition and terminologies, threaded binary tree, BST.	1
	2. Avl tree, balance multi way search tree: 2-3 tree, red- black tree, B tree, B+ tree, tries,spatial data representation using k-d tree, quad tree.	3
3	**Algorithm Design Techniques:**	**15L**
	1. Divide and Conquer: Basic method, use, Examples – Binary Search, Merge Sort ,Quick Sort and there complexity.	2
	2. Priority Queue:Definition, Heap Sort and its complexity.	1
	3. Dynamic Programming:Basic method, use, Examples – Matrix Chain Manipulation it's complexity;All pair shortest paths(Floyd-Warshall algorithm), single source shortest path(Bellman-ford algorithm), Travelling Salesman Problemand their complexities	4
	4. Backtracking:Basic method, use, Examples – 8 queens problem,Graph colouring problem.	1
	5. Branch and Bound:15 puzzle problem and its applications.	1
	6. Greedy Method:Basic method, use, Examples – Knapsack problem, Job sequencing with deadlines, Activity selection problem,single source shortest path (Dijkstra algorithm), Minimum cost spanning tree by Prim's and Kruskal's algorithm, their complexities	3
	7. Graph traversal algorithm: Breadth First Search (BFS) and Depth First Search	1

	(DFS) – complexity and comparison with different edges	
	8. Disjoint set manipulation:Set manipulation algorithm like UNION-FIND, union by rank, path compression	1
	9. Graph traversal algorithm:Breadth First Search (BFS) and Depth First Search (DFS) – complexity and comparison with different edges	1
4	**Computational geometry:**	**8L**
	1. Robust geometric primitives, convex hull, triangulation, voronoi diagrams, nearest neighbor search, range search, point location, intersection detection, bin packing,	4
	2. Medial-axis transform, polygon partitioning, simplifying polygons, shape similarity, motion planning, maintaining line arrangements, minkowski sum.	4
5	**Set and string problems:**	**5L**
	1. set cover, set packing, string matching, approximate string matching, text compression, cryptography, finite state machine minimization	3
	2. longestCommon substring/subsequence, shortest common superstring.	2
6	**Amortized Analysis:**	**1L**
	1. Aggregate, Accounting, and Potential Method.	1
7	**Advanced areas:**	**10L**
	1. P class, NP class, NP hard class, NP complete class	1
	2. Their interrelationship, Satisfiability problem, Cook's theorem (Statement only), Clique decision problem, vertex cover problem, Hamiltonian cycle problem	2
	3. Necessity of approximation scheme, performance guarantee, polynomial time approximation schemes,	2
	4. Vertex Cover problem, travelling salesman problem	1
	5. Randomized algorithms, multithreaded algorithms, parallel algorithms.	4
Total Number Of Hours = 45		

Faculty In-Charge HOD, CSE Dept.

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Assignment:

Module-1(Complexity analysis):

1. State the master theorem and prove it.
2. Solve the recurrence relation:
   a)$T(n)=4T(n-1)+n^{1/2}$
   b)$T(n)=2T(n^{1/2})+n\log n$

Module-2 (Advanced data structure):

1. Describe the insertion and deletion operation od red black tree and why it is required.
2. Define k-d tree and quad tree

Module-3(Algorithm Design Techniques):

1. Write down the algorithm of ternarysearch and find out the time complexity.
2. ModifytheMerge sortalgorithmsothattheinput array AisdividedintoKparts instead of2.Analyzeyouralgorithm.AssumeK>1.
3. Write analgorithmtosortanelementinaascendingorder using min-heap and analyse it.
4. Findoutthetimecomplexityofheapify function in heap sort.
5. Explain Floyd warshall algorithm with example.
6. Considerthefollowingfivematrices:
   A1=2X3,A2=3X4,A3=4X6,A4=6X2,A5=2X7.
   (i) How manyparenthesizationarepossibletomultiplythese matrices?
   (ii) Give aparenthesizedexpressionfor the order in which this optimalnumberof multiplicationsisachieved.
   (iii) Findtheoptimalcostofthesolution
7. Write down the complexity of N queens problem.
8. Draw the state space tree of 4coloring problem.
9. Find out the time complexity of prime algorithm and dijkstra algorithm.
10. What are the features are present in any greedy algorithm?
11. Define path compression technique in union and find algorithm
12. Howprims algorithm follow this method explain with example
13. Find out the time complexity of BFS.
14. Write an algorithm to find the graph contain any back edge or not

Module-12(Computational geometry):

1. The convex hull of a set S is defined to be the intersection of all convex sets that contain S. For the convex hull of a set of points it was indicated that the convex hull is the convex set with smallest perimeter. We want to show that these are equivalent definitions. a. Prove that the intersection of two convex sets is again convex. This implies that the intersection of a finite family of convex sets is convex as well. b. Prove that the smallest perimeter polygon P containing a set of points P is convex. c. Prove that any convex set containing the set of points P contains the smallest perimeter polygon P.
2. Verify that the algorithm CONVEXHULL with the indicated modifications correctly computes the convex hull, also of degenerate sets of points. Consider for example such nasty cases as a set of points that all lie on one (vertical) line.

Module-12 (Set and string problems):

1. Explain KMP algorithm with example.Why KMP is better than Naïve and string matching with finite automata.
2. Write an algorithm to find out the longest common sub sequence in a string.

Module-13(Amortize analysis):

1. Define Aggregate Method Accounting Method and Potential Method**.**

Module-15(Advanced areas:):

1. Show all satisfiability problem are verifiable in polynomial time.
2. Prove that approx-vertex-cover is 2-approximation algorithm
3. Write a short note on Randomized algorithms, multithreaded algorithms, parallel algorithms.

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: Artificial Neural Network **Subject Code- MCS105A**

Year: 1st Year **Semester: First**

Module Number	Topics	Number of Lectures
1	**Introduction to artificial neural networks :**	**5L**
	1. Biological neural networks, Pattern analysis tasks: Classification, Regression, Clustering.	2
	2. Computational models of neurons.	1
	3. Structures of neural networks	1
	4. Learning principles	1
2	**Linear models for regression and classification:**	**8L**
	1. Polynomial curve fitting, Bayesian curve fitting.	2
	2. Linear basis function models, Bias-variance decomposition.	2
	3. Bayesian linear regression, Least squares for classification.	2
	4. Logistic regression for classification, Bayesian logistic regression for classification.	2
3.	**Feed forward neural networks :**	**8L**
	1. Pattern classification using perceptron, Multilayer feed forward neural networks (MLFFNNs).	2
	2. Pattern classification and regression using MLFFNNs.	**2**
	3. Error back propagation learning.	**1**
	4. Fast learning methods: Conjugate gradient method, Auto associative neural networks.	**2**
	5. Bayesian neural networks.	**1**
4	**Radial basis function networks :**	**5L**
	1. Regularization theory.	2
	2. RBF networks for function approximation.	2
	3. RBF networks for pattern classification.	1
5	**Self-organizing maps :**	**4L**
	1. Pattern clustering, Topological mapping	2
	2. Kohonen's self-organizing map.	2
6	**Feedback neural networks :**	**5L**
	1. Pattern storage and retrieval, Hopfield model	2
	2. Boltzmann machine	1

	3. Recurrent neural networks.	2
7	**Kernel methods for pattern analysis :**	**8L**
	1. Statistical learning theory	2
	2. Support vector machines for pattern classification	2
	3. Support vector regression for function approximation	2
	4. Relevance vector machines for classification and regression	2
Total Number Of Hours = 43		

Faculty In-Charge HOD, CSE Dept.

Assignment:

Module-1(Introduction):
1. Write Down short Notes: Classification & Regrassion.
2. Draw the structure of Neural networks & describe it.

Module-2(Linear models for regression and classification):
1. Notes: Bayesian linear regression & Bias-variance decomposition.
2. Prove that: MSE = Bias^2 + Var.

Module-3(Feed forward neural networks):
1. Short Notes: Bayesian neural networks.
2. Describe the Error back propagation learning.
3. Write algorithm of back-propagation rule

Module-4(Radial basis function networks):
1. Write down short notes**:** RBF networks.

Module-5(Self-organizing maps):
1. Consider a Kohonen net with two cluster (outputs) units & five input units. The weight vectors for the output units are W1=[1,0.8,0.6,0.4,0.2] and W2=[1,0.5,1,0.5,1]. Use the square of the Euclidean distance to find the winning neuron for the input pattern X=[0.5,1,0.5,0,0.5]. Find the new weights for the winning unit. Assume learning rate as 0.2.

Module-6(Feedback neural networks):
1. Short Notes: Boltzmann machine & Recurrent neural networks.

Module-7(Kernel methods for pattern analysis):
1. Write down short note for SVM.
2. How its works in classification & regression?

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: Agent Based Intelligent Systems **Subject Code-MCSE105B**
Year: 1st Year **Semester: First**

Module Number	Topics	Number of Lectures
1	**Introduction:**	**9L**
	1. Definitions, Foundations, History, Intelligent Agents	3L
	2. Problem Solving, Searching, Heuristics	3L
	3. Constraint Satisfaction Problems, Game playing.	3L
2	**Knowledge Representation And Resoning:**	**9L**
	1. Logical Agents, First order logic, First Order Inference	3L
	2. Unification, Chaining, Resolution	2L
	3. Strategies, Knowledge Representation	3L
	4. Objects, Actions, Events.	1L
3	**Planning Agent:**	**9L**
	1. Planning Problem, State Space Search	2L
	2. Partial Order Planning, Graphs, Nondeterministic Domains	3L
	3. Conditional Planning, Continuous Planning, Multi-Agent Planning.	4L
4	**Agent And Uncertainty:**	**9L**
	1. Acting under uncertainty – Probability Notation, Bayes Rule and use	3L
	2. Bayesian Networks, Other Approaches	2L
	3. Time and Uncertainty, Temporal Models	2L
	4. Utility Theory, Decision Network – Complex Decisions.	2L
5	**Higher Level Agent:**	**9L**
	1. Knowledge in Learning, Relevance Information	2L
	2. Statistical Learning Methods, Reinforcement Learning	3L
	3. Communication, Formal Grammar, Augmented Grammars, Future of AI.	4L

Faculty In-Charge HOD, CSE Dept.

Assignment:
Module-1(Introduction):

1. Intelligent Agents
2. Constraint Satisfaction Problems and Game playing.

Module-2 (Knowledge Representation And Resoning):

1. Logical Agents
2. First order logic and First Order Inference

Module-3(Planning Agent):

1. State Space Search
2. Different planning

Module-4(Agent And Uncertainty)

1. Probability Notation
2. Bayesian Networks

Module-5(Higher Level Agent):

1. Reinforcement Learning

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: Advanced Soft Computing **Subject Code-MCSE105C**

Year: 1st Year **Semester: First**

Module Number	Topics	Number of Lectures
1	**Introduction:**	**3L**
	1. Introduction to Soft Computing; Difference between Hard and Soft Computing;	1
	2. Introduction to Fuzzy Systems, Artificial Neural Network, Evolutionary Algorithms, Rough Set Theory; Hybrid Systems	2
2	**Fuzzy Sets & Logic**	**10L**
	1. Introduction to Fuzzy Sets, Classical and Fuzzy Sets.	1
	2. Fuzzy Sets - Membership Function, Basic Operations, Linguistic Variable, Properties.	2
	3. Fuzzy relations - Cartesian product, Operations on relations; Crisp logic—Laws of propositional logic, Inference.	2
	4. Predicate logic—Interpretations, Inference, Fuzzy logic—Quantifiers, Inference.	2
	5. Fuzzy Rule based system; De-fuzzification methods; Basic Applications of Fuzzy Sets and Logics.	3
3	**Pattern Recognition**	**3L**
	1. Pattern Classification, Pattern Association, Clustering.	1
	2. Simple Clustering algorithm, k-means & k-medoid based algorithm.	2
4	**Artificial Neural Network**	**13L**
	1. Neural Networks: Introduction.	1
	2. Mathematical Models, ANN architecture.	1
	3. Learning rules, Supervised, Unsupervised and reinforcement Learning.	2
	4. Multilayer Perception, Applications of Artificial Neural Networks.	2
	5. Competitive learning networks, Kohonen self organizing networks, Hebbian learning.	3
	6. Hopfield Networks, Associative Memories, The boltzman machine; Applications of ANN	4
5	**Genetic Algorithms**	**10L**
	1. Introduction, Single and Multi-Objective Optimization, Encoding, Fitness Function, Genetic Operations, Genetic Parameters; Genetic Algorithm; Basic Applications.	5
	2. Schema theorem; Convergence Theory; Multiobjective optimization using GA	5

	(MOGA); Non-Dominated Sorting	
6	**Hybrid Systems**	**6L**
	1. Hybrid systems, GA based ANN (Optimal Weight Selection).	3
	2. Neuro Fuzzy Systems, fuzzy Neuron, architecture, learning, application.	3

Faculty In-Charge HOD, CSE Dept.

Assignment:

Module-1(Introduction):

1. Difference between Hard and Soft Computing.
2. Artificial Neural Network
3. Evolutionary Algorithms
4. Rough Set Theory
5. Hybrid Systems

Module-2 (Fuzzy Sets & Logic):

1. Classical and Fuzzy Sets
2. Crisp logic—Laws of propositional logic.
3. Predicate logic
4. Basic Applications of Fuzzy Sets and Logics.

Module-3(Pattern Recognition):

1. k-means & k-medoid based algorithm.

Module-4(Artificial Neural Network)

1. Supervised, Unsupervised and reinforcement Learning
2. Multilayer Perception
3. Hopfield Networks

Module-5(Genetic Algorithms):

1. Genetic Algorithm.
2. Convergence Theory

Module-6(Hybrid Systems):

1. Fuzzy Neuron

UNIVERSITY OF ENGINEERING AND MANAGEMENT,JAIPUR

Lecture-wise Plan

Subject Name: Object Oriented Information System Design Subject Code-MCSE105D

Year:1st Year Semester: 1st

Module Number	Topics	Number of Lectures
1	**Data and Information**	**3L**
	1. Types of information: operational, tactical, strategic and statutory –	1
	2. why do we need information systems – management structure – requirements of information at different levels of management – functional allocation of management	1
	3. requirements of information for various functions – qualities of information – small case study	1
2	**Systems Analysis and Design Life Cycle**	**3L**
	1. Requirements determination – requirements specifications	1
	2. feasibility analysis – final specifications – hardware and software study – system design – system implementation	1
	3. System evaluation – system modification. Role of systems analyst – attributes of a systems analyst – tools used in system analysis	1
3	**Information**	**3L**
	1. Information gathering	1
	2. strategies – methods – case study – documenting study – system requirements specification	1
	3. From narratives of requirements to classification of requirements as strategic, tactical, operational and statutory. Example case study	1
4	**Feasibility analysis**	**6L**
	1. Feasibility analysis – deciding project goals – examining alternative solutions – cost	2
	2. benefit analysis – quantifications of costs and benefits – payback period – system proposal preparation for managements	2
	3. parts and documentation of a proposal – tools for prototype creation	2
		2
5	**Tools for systems analysts**	**3L**
	1. data flow diagrams –	1
	2. case study for use of DFD, good conventions – leveling of DFDs – leveling rules	1
	3. logical and physical DFDs – software tools to create DFDs .	1

6	**Structured systems analysis and design**	**4L**
	1. Structured systems analysis and design – procedure specifications in structured English – examples and cases 2. decision tables for complex logical specifications – specification oriented design vs procedure oriented design	2 2
7	**Data oriented systems design**	**6L**
	1. entity relationship model – E-R diagrams 2. relationships cardinality and participation – normalizing relations 3. various normal forms and their need – some examples of relational data base design	2 2 2
8	**Data input methods**	**3L**
	1. coding techniques 2. requirements of coding schemes – error detection of codes 3. validating input data – input data controls interactive data input	1 1 1
Total Number Of Hours = 31		

Faculty In-Charge HOD, CSE Dept.

UNIVERSITY OF ENGINEERING AND MANAGEMENT,JAIPUR

Lecture-wise Plan

Subject Name: Object Oriented Information System Design Subject Code-MCSE105D
Year:1st Year Semester: 1st
Assignment:

Module-I:

1. Why do we need information systems? Explain management structure.
2. Explain the requirements of information at different levels of management.

Module-II:

1.Explain Requirements determination and requirements specifications with proper example.

2. What do you mean by feasibility analysis? Explain with proper example.

Module-III:

1. ExplainInformation gathering strategies.
2. What do you mean by system requirements specification?

Module-IV:

1. Explain Feasibility analysis.
2. What is payback period?

Module-V:

1. What is DFD?
2. Explain logical and physical DFDs.

Module-VI:

1. What isstructured systems analysis and design?
2. Differentiate specification oriented design and procedure oriented design system.

Module-VII:

1. What do you mean by E-R diagram? Explain with an example.
2. What is relationships cardinality?

Module-VIII:

1. Explain the requirements of coding schemes. Why error detection of codes is required?
2. How you validate the inputted data?

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: Software Engineering &CASE tools **Subject Code-MCSE105E**
Year: 1st Year **Semester: First**

Module Number	Topics	Number of Lectures
1	**Software Engineering**	**6L**
	1. Objectives, Definitions	1
	2. Software Process models - Waterfall Model , Prototype model, RAD	3
	3. Evolutionary Models	1
	4. Incremental, Spiral.	1
2	**Software Project Planning**	**4L**
	1. Feasibility Analysis, Technical Feasibility	2
	2. Cost- Benefit Analysis,	1
	3. COCOMO model.	1
3.	**Structured Analysis**	**5L**
	1. Context diagram and DFD, Physical and Logical DFDs.	3
	2. Data Modelling, ER diagrams, Software Requirements Specification.	2
4	**Design Aspects**	**4L**
	1. Top-Down And Bottom-Up design; Decision tree, decision table and structured English	2
	2. Structure chart, Transform analysis Functional vs. Object- Oriented approach.	2
5	**Unified Modelling Language**	**6L**
	1. Class diagram, interaction diagram: state chart diagram,	2
	2. collaboration diagram, sequence diagram,	2
	3. Activity diagram, implementation diagram.	2
6	**Coding & Documentation**	**6L**
	1. Structured Programming, Modular Programming, Module Relationship- Coupling,	3
	2. Cohesion, OO Programming, Information Hiding, Reuse, System Documentation.	3
7	**Testing**	**4L**
	1. Levels of Testing	1
	2. Integration Testing	1
	3. System Testing.	1
	Software Quality	**4L**
	1. Quality Assurance	1

8	**2.** Software Maintenance	1
	3. Software Configuration Management	1
	4. Software Architecture, Computer Aided Software Engineering (CASE) tool	2
9	**Object modeling and design**	**10**
	Classes, objects, relationships, key abstractions, common mechanisms, diagrams, class diagrams, advanced classes, advanced relationships,	**3**
	interfaces, types, roles, packages, instances, object diagrams, interactions, use cases, use case diagrams, interaction diagrams, activity diagrams, events and signals, state machines,	**3**
	processes, threads, state chart diagrams, components, deployment, collaborations, patterns and frameworks, component diagrams, systems and models, code generation and reverse engineering.	**6**
Total Number Of Hours = 49		

Faculty In-Charge HOD, CSE Dept.

Assignment:

Module-1(Software Engineering):

1. Explain Software Engineering as a layered technology with a neat sketch.
2. What are process models why do we require them? Explain in detail any one of the process model?
3. Why it is important to design a software and explain the role it play?

Module-2 (Software Project Planning):

1. What are the purposes of Data Flow diagrams, Entity-Relationship diagrams? Give an example diagram of each. (10 mks)

Module-3(Structured Analysis):

1. What is functional and non-functional requirements?
2. What is a requirement modeling? Explain about the types of requirement modeling
3. How can software project estimation be done by empirical estimation model

Module-4(Design Aspects):

1. How designing is done by functional based component design?

Module-5(Unified Modelling Language):

1. Give a brief description about class hierarchies and class based component design?

Module-6(Coding & Documentation):

Module-7(Testing):

1. What is user acceptance testing? Explain different testings in user acceptance testing. Why is it necessary?

Module-8(Software Quality):

1. Explain about the architecture of software and its importance?

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lecture-wise Plan

Subject Name: Computer Graphics — Subject Code-CS604B

Year: 3rd Year — Semester: Sixth

Module Number	Course Details	Number of Lectures
UNIT 1	**Introduction to computer graphics & graphics systems:**	**4LH**
1	• Overview of computer graphics, representing pictures, preparing, presenting & interacting with pictures for presentations • Visualization & image processing; RGB color model, direct coding, lookup table • storage tube graphics display, Raster scan display, 3D viewing devices, Plotters, printers, digitizers, Light pens etc. • Active & Passive graphics devices	
2	**Scan conversion:** • Points & lines, Line drawing algorithms • DDA algorithm, Bresenham's line algorithm, Circle generation algorithm • Ellipse generating algorithm; scan line polygon, fill algorithm • boundary fill algorithm, flood fill algorithm.	**6LH**
UNIT 2	**2D transformation & viewing:**	**7LH**
3	• Basic transformations: translation, rotation, scaling; • Matrix representations & homogeneous coordinates, transformations between coordinate systems; • Reflection shear; Transformation of points, lines, parallel lines, intersecting lines. • Cohen and Sutherland line clipping, Sutherland-Hodgeman Polygon clipping, Cyrus-beck clipping method	
4	**3D transformation & viewing:** • 3D transformations: translation, rotation, scaling & other transformations • Rotation about an arbitrary axis in space • Clipping, view port clipping, 3D viewing.	**7LH**
UNIT 3	**Curves and Hidden surfaces:**	

5	• Curve representation, surfaces, designs, Bezier curves • B-spline curves, end conditions for periodic B-spline curves, rational B-spline curves. • Depth comparison, Z-buffer algorithm, Back face detection • BSP tree method, the Painter's algorithm, scan-line algorithm • Hidden line elimination, wire frame methods , fractal - geometry.	**7LH**
6	**Introduction to Ray-tracing:** • Human vision and color • Lighting, Reflection and transmission models.	**8LH**
UNIT 4 **7**	**Multimedia:** • Introduction to Multimedia: Concepts, uses of multimedia, hypertext and hypermedia • Image, video and audio standards. Audio: digital audio, MIDI, processing sound, sampling, compression • Video: MPEG compression standards, compression through spatial and temporal redundancy, inter-frame and intra-frame compression. Animation: types, techniques, key frame animation, utility, morphing • Virtual Reality concepts.	**7LH**
	Total Number Of Hours = 44	

Faculty In-Charge HOD, CSE Dept.

Assignment:

Module-1(Introduction to computer graphics & graphics systems):

1. A monochromatic graphic display system has 525 scan lines with an aspect ratio 9:16. If each pixel is displaceable in 512 shades
   (i) How many pixels are displayed on the screen?
   (ii) What is the picture storage memory size?
2. What do you mean by window and viewport? Describe the relationship for window to viewport mapping.

Module-3 (2D transformation & viewing):

1. Prove that successive scaling is multiplicative
2. Write down mid-point ellipse drawing algorithm

3. a rectangular 2D clipping window has its lower left corner at (100,10) and upper right corner at (160,40). Find visible portion of lines A(50,0), B(120,30) and C(120,20), D(140,80) using mid point sub division algorithm.

Module-4 (Curves and Hidden surfaces)

1. Write down the procedure for drawing B-spline curves and also write down its property
2. Derive the condition to be satisfied when joining two Bezier curves with second order continuity at the joints.

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lab Manual

Title of Course: Operating System Lab
Course Code: MCSE191
L-T-P scheme: 0-0-3 **Course Credit: 2**

Objectives:

1. To learn and understand system calls (remote procedure calls) related to files, processes, threads, signals, semaphores and implement system programs based on that.
2. To provide an understanding of the design aspects of operating system.
3. To provide an efficient understanding of the language translation peculiarities by designing a complete translator for a mini language.

Learning Outcomes: The students will have a detailed knowledge of the concepts of process and shared memory, aware of a variety of approaches to process management and main-memory management, including interference, deadlock, scheduling, fragmentation, thrashing, learn the basics behind file systems and input output systems and understand the fundamentals of network and distributed operating systems. Upon the completion of Operating Systems practical course, the student will be able to:

- **Understand** and implement basic services and functionalities of the operating system using system calls.
- **Use** modern operating system calls and synchronization libraries in software/ hardware interfaces**.**
- **Understand** the benefits of thread over process and implement synchronized programs using multithreading concepts.
- **Analyze** and simulate Deadlock Avoidance and Protection algorithm like Bankers.
- **Implement** memory management schemes
- **Implement** remote procedure call
- **Understand** producer and consumer problem.

Course Contents:

Exercises that must be done in this course are listed below:

Exercise No.1: Simulate Banker's Algorithm for Dead Lock Avoidance
Exercise No.2: Simulate Banker's Algorithm for Dead Lock Prevention
Exercise No. 3: Simulate Paging Technique of Memory Management
Exercise No. 4: Thread Creation
Exercise No. 5: Process Creation
Exercise No. 6: Producer and Consumer Problem
Exercise No. 7: Implementation of Remote Procedure Call

Text Book:

1. Maurice J. Bach, Design of the UNIX Operating System, PHI.

Recommended Systems/Software Requirements:

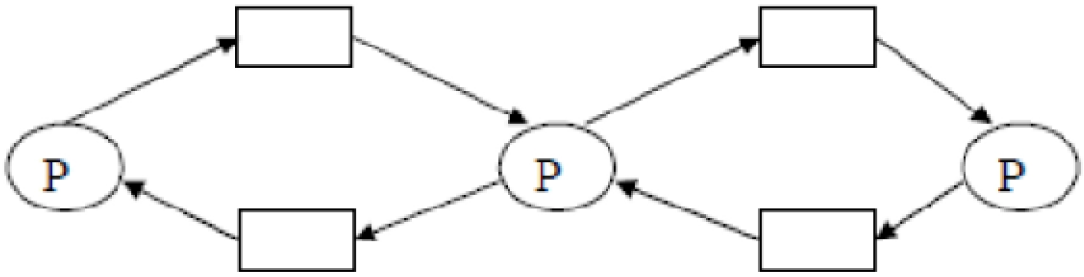
1. Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
2. Turbo C or TC3 complier in Windows XP or Linux Operating System.

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

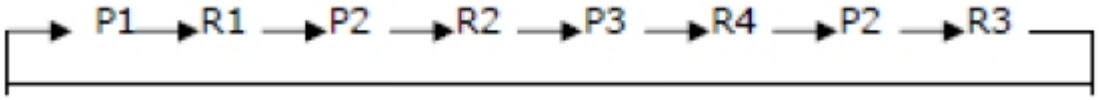
Lab Manual

Experiment No: 1 BANKER'S DEADLOCK AVOIDANCE
Aim: To Simulate Bankers Algorithm for Deadlock Avoidance.
Description:

Deadlock: A process request the resources, the resources are not available at that time, so the process enter into the waiting state. The requesting resources are held by another waiting process, both are in waiting state, this situation is said to be Deadlock. A deadlocked system must satisfied the following 4 conditions. These are:
(i) Mutual Exclusion: Mutual Exclusion means resources are in non-sharable mode only, it means only one process at a time can use a process.
(ii) Hold and Wait: Each and every process is the deadlock state, must holding at least one resource and is waiting for additional resources, that are currently being held by another process.



(iii) No Preemption: No Preemption means resources are not released in the middle of the work, they released only after the process has completed its task.
(iv) Circular Wait: If process P1 is waiting for a resource R1, it is held by P2, process P2 is waiting for R2, R2 held by P3, P3 is waiting for R4, R4 is held by P2, P2 waiting for resource R3, it is held by P1.



Deadlock Avoidance: It is one of the method of dynamically escaping from the deadlocks. In this scheme, if a process request for resources, the avoidance algorithm checks before the allocation of resources about the state of system. If the state is safe, the system allocate the resources to the requesting process otherwise (unsafe) do not allocate the resources. So taking care before the allocation said to be deadlock avoidance.
Banker's Algorithm: It is the deadlock avoidance algorithm, the name was chosen because the bank never allocates more than the available cash.
Available: A vector of length 'm' indicates the number of available resources of each type. If available[j]=k, there are 'k' instances of resource types Rj available.
Allocation: An nxm matrix defines the number of resources of each type currently allocated to each process. If allocation [i,j]=k, then process Pi is currently allocated 'k' instances of resources type Rj.
Max: An nxm matrix defines the maximum demand of each process. If max [i,j]=k, then Pi may request at most 'k' instances of resource type Rj.
Need: An nxm matrix indicates the remaining resources need of each process. If need [I,j]=k, then Pi may need 'k' more instances of resource type Rj to complete this task. There fore, Need[i,j]=Max[i,j]-Allocation[I,j]

Safety Algorithm:
1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
   Finish[i] =False
   Need<=Work

   If no such I exist go to step 4.
3. Work=work Allocation, Finish[i] =True;

4. if Finish[1]=True for all I, then the system is in safe state.

Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2. Otherwise raise an error condition.
2. if Request<=Available go to step 3. Otherwise Pi must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;

Available=Available-Request I;
Allocation I =Allocation +Request I;
Need i=Need i-Request I;

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However, if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

Algorithm for Banker's Deadlock Avoidance:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. or not if we allow the request.
10. stop the program.

/* Program to Simulate Bankers Algorithm for Dead Lock Avoidance */

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
int alloc[10][10],max[10][10];
int avail[10],work[10],total[10];
int i,j,k,n,need[10][10];
int m;
int count=0,c=0;
char finish[10];
clrscr();
printf("Enter the no. of processes and resources:");
scanf("%d%d",&n,&m);
for(i=0;i<=n;i++)
finish[i]='n';
printf("Enter the claim matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&max[i][j]);
printf("Enter the allocation matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&alloc[i][j]);
printf("Resource vector:");
for(i=0;i<m;i++)
scanf("%d",&total[i]);
for(i=0;i<m;i++)
```

```
for(i=0;i<n;i++)
for(j=0;j<m;j++)
avail[j]+=alloc[i][j];
for(i=0;i<m;i++)
work[i]=avail[i];
for(j=0;j<m;j++)
work[j]=total[j]-work[j];
for(i=0;i<n;i++)
for(j=0;j<m;j++)
need[i][j]=max[i][j]-alloc[i][j];
A:for(i=0;i<n;i++)
{
c=0;
for(j=0;j<m;j++)
if((need[i][j]<=work[j])&&(finish[i]=='n'))
c++;
if(c==m)
{
printf("All the resources can be allocated to Process %d",
i+1);
printf("\n\nAvailable resources are:");
for(k=0;k<m;k++)
{
work[k]+=alloc[i][k];
printf("%4d",work[k]);
}
printf("\n");
finish[i]='y';
printf("\nProcess %d executed?:%c \n",i+1,finish[i]);
count++;
}
}
if(count!=n)
goto A;
else
printf("\n System is in safe mode");
printf("\n The given state is safe state");
getch();
}
```

OUTPUT 1:

```
Enter the no. of processes and resources: 4 3
Enter the claim matrix:
    3 2 2
    6 1 3
    3 1 4
    4 2 2
Enter the allocation matrix:
    1 0 0
    6 1 2
    2 1 1
    0 0 2
    Resource vector:9 3 6
    All the resources can be allocated to Process 2
    Available resources are: 6 2 3
    Process 2 executed?:y
    All the resources can be allocated to Process 3
```

```
Avaiable resources are: 8 3 4
Prcess 3 executed?:y
All the resources can be allocated to Process 4
Avilable resources are: 8 3 6
Prcess 4 executed?:y
All the resources can be allocated to Process 1
Aailable resources are: 9 3 6
Prcess 1 executed?:y
System is in safe mode
The given state is safe state
```

Experiment No:2 BANKER'S DEADLOCK PREVENTION
Aim: To Simulate Bankers Algorithm for Deadlock Prevention.

/* Program to Simulate Bankers Algorithm for Dead Lock Prevention */

```
#include<stdio.h>
#include<conio.h>
void main()
{
char job[10][10];
int time[10],avail,tem[10],temp[10];
int safe[10];
int ind=1,i,j,q,n,t;
clrscr();
printf("Enter no of jobs: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter name and time: ");
scanf("%s%d",&job[i],&time[i]);
}
printf("Enter the available resources:");
scanf("%d",&avail);
for(i=0;i<n;i++)
{
temp[i]=time[i];
tem[i]=i;
}
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
{
if(temp[i]>temp[j])
{
t=temp[i];
temp[i]=temp[j];
temp[j]=t;
t=tem[i];
tem[i]=tem[j];
tem[j]=t;
}
}
for(i=0;i<n;i++)
{
q=tem[i];
if(time[q]<=avail)
{
```

```
avail=avail-tem[q];
//printf("%s",job[safe[ind]]);
ind++;
}
else
{
printf("No safe sequence\n");
}
}
printf("Safe sequence is:");
for(i=1;i<ind; i++)
printf(" %s %d\n",job[safe[i]],time[safe[i]]);
getch();
}
```

OUTPUT 1:

```
Enter no of jobs:4
Enter name and time: A 1
Enter name and time: B 4
Enter name and time: C 2
Enter name and time: D 3
Enter the available resources: 20
Safe sequence is: A 1, C 2, D 3, B 4
```

Experiment No: 3 PAGING TECHNIQUE OF MEMORY MANAGEMENT

AIM: To implement the Memory management policy- Paging.

Description: Paging is an efficient memory management scheme because it is noncontiguous memory allocation method. The basic idea of paging is the physical memory (main memory) is divided into fixed sized blocks called frames, the logical address space is divided into fixed sized blocks, called pages, but page size and frame size should be equal. The size of the frame or a page is depending on operating system. In this scheme the operating system maintains a data structure that is page table; it is used for mapping purpose. The page table specifies the some useful information; it tells which frames are there and so on. The page table consisting of two fields, one is the page number and other one is frame number. Every address generated by the CPU divided into two parts; one is page number and second is page offset or displacement. The pages are loaded into available free frames in the physical memory.

Algorithm for Paging Technique:

Step 1: Read all the necessary input from the keyboard.
Step 2: Pages - Logical memory is broken into fixed - sized blocks.
Step 3: Frames – Physical memory is broken into fixed – sized blocks.
Step 4: Calculate the physical address using the following
Physical address = (Frame number * Frame size) + offset
Step 5: Display the physical address.
Step 6: Stop the process.

/* Program to simulate paging technique of memory management */

```
#include<stdio.h>
void main()
{
int p, ps, i;
int *sa;
clrscr();
printf("Enter how many pages: ");
scanf("%d",&np);
printf("Enter page size: ");
```

```
scanf("%d",&ps);
for(i=0;i< np;i++)
{
sa[i]=(int)malloc(ps);
printf("Page %d address is %d\n", i, sa[i]);
}
getch();
}
```

OUTPUT 1:

```
Enter how many pages: 5
Enter page size: 4
Page 0 address is 3080
Page 1 address is 3088
Page 2 address is 3096
Page 3 address is 3104
Page 4 address is 3112
```

OUTPUT 2:

```
Enter how many pages: 7
Enter page size: 8
Page 0 address is 3080
Page 1 address is 3096
Page 2 address is 3112
Page 3 address is 3128
Page 4 address is 3144
Page 5 address is 3160
Page 6 address is 3176
```

Experiment No: 4(a) THREAD CREATION

Aim: Design ,develop and execute a program using any thread library to create number of threads specified by the user ,each thread independently generate a random integer as an upper limit and then computes and prints the number of primes less than or equal to that upper limit along with that upper limit.

DESCRIPTION:

What is a Thread?

- Technically, a thread is defined as an independent stream of instructions that can be scheduled to run as such by the operating system. But what does this mean?
- To the software developer, the concept of a "procedure" that runs independently from its main program may best describe a thread.

What are Pthreads?

- Pthreads are defined as a set of C language programming types and procedure calls, implemented with a pthread.h header/include file and a thread library - though this library may be part of another library, such as lab, in some implementations.

Why Pthreads?

- In the world of high performance computing, the primary motivation for using Pthreads is to realize potential program performance gains.
- When compared to the cost of creating and managing a process, a thread can be created with much less operating system overhead. Managing threads requires fewer system resources than managing processes.
- All threads within a process share the same address space. Inter-thread communication is more efficient and in many cases, easier to use than inter-process communication.

The Pthreads API:

The original Pthreads API was defined in the ANSI/IEEE POSIX 1003.1 - 1995 standard. The POSIX standard has continued to evolve and undergo revisions, including the Pthreads specification. The subroutines which comprise the Pthreads API can be informally grouped into four major groups:

1. **Thread management:** Routines that work directly on threads - creating, detaching, joining, etc. They also include functions to set/query thread attributes (joinable, scheduling etc.)
2. **Mutexes:** Routines that deal with synchronization, called a "mutex", which is an abbreviation for "mutual exclusion". Mutex functions provide for creating, destroying, locking and unlocking mutexes.
3. **Condition variables:** Routines that address communications between threads that share a mutex. Based upon programmer specified conditions. Functions to set/query condition variable attributes are also included.
4. **Synchronization:** Routines that manage read/write locks and barriers.

Compiling Threaded Programs:

- Several examples of compile commands used for pthreads codes are listed in the table.

Compiler / Platform	Compiler Command	Description
INTEL Linux	**icc –pthread**	C
	icpc –pthread	C++
PGI Linux	**pgcc –lpthread**	C
	pgCC –lpthread	C++
GNU	**gcc –pthread**	GNU C

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR
Lab Manual

Linux, Blue Gene	**g++ -pthread**	GNU C++

Thread management:

pthread_create(thread, attar, start routine, rag):

- Initially, your main () program comprises a single, default thread. All other threads must be explicitly created by the programmer.
- Pthread_create creates a new thread and makes it executable. This routine can be called any number of times from anywhere within your code.
- pthread_create arguments:

  - ✓ Thread: An opaque, unique identifier for the new thread returned by the subroutine.
  - ✓ Attr: An opaque attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values.
  - ✓ start_routine: the C routine that the thread will execute once it is created.
  - ✓ Arg: A single argument that may be passed to *start_routine*. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

pthread_exit (): There are several ways in which a thread may be terminated:

- The thread returns normally from its starting routine. Its work is done.
- The thread makes a call to the pthread_exit subroutine - whether its work is done or not.
- The entire process is terminated due to making a call to either the exec() or exit()
- If main() finishes first, without calling pthread_exit explicitly itself.

Joining: pthread_join (threadid, status)

"Joining" is one way to accomplish synchronization between threads.

- The pthread_join () subroutine blocks the calling thread until the specified threadid thread terminates.
- The programmer is able to obtain the target thread's termination return status if it was specified in the target thread's call to pthread_exit ().
- A joining thread can match one pthread_join () call. It is a logical error to attempt multiple joins on the same thread.

Detaching: The pthread_detach () routine can be used to explicitly detach a thread even though it was created as joinable.

pthread_self (): Pthread_self returns the unique, system assigned thread ID of the calling thread.

PROGRAM CODE:

```
#include<stdio.h>
#include<pthread.h>
#include<math.h>
#include<stdlib.h>
Pthread_t ntip [50];

int isprime (int n)
{
    int i,cnt=0;
    if(n==2)
    {
            printf("\n %d ",n);
            return 0;
    }
    for(i=2;i<n;i++)
    {
            if(n%i==0)
            return 1;
    }
    printf("\n%d",n);
    return 0;
}

void prime_disp(int r)
{
    int i,cnt=0;
    printf(" \n In thread %u \n",(unsigned int)pthread_self());
    printf("\n Prime number list :\n");
    for(i=2;i<=r;i++)
    {
            if(!isprime(i))
            cnt++;
    }
    printf("\n no of prime number in the limit is %d\n",cnt);
}

void * thr_fn()
{
    printf("thread.....%u\n",(unsigned int)pthread_self());
    int r=rand();
    //printf(" \n In thread %d \n",(unsigned int)pthread_self());
    printf("Random number obtained = %d\n",r%50);
    prime_disp(r%50);
    return((void *)1);
```

```
}

int main()
{
    int err,i,n;
    void *tret;
    printf("Enter number of thread(upto 50) : ");
    scanf("%d",&n);
    for(i=0;i<n;i++) {
    err=pthread_create(&ntip[i],NULL,thr_fn,NULL);
            if(err!=0)
            {
                    printf("Error");
                    exit(0);
            }
    }
    //sleep(5);
    for(i=0;i<n;i++)
    pthread_join(ntip[i],&tret);
    printf("\n main thread\n ");
    return 0;
}
```

Output:

```
root@localhost student]# gcc -lpthread pp1.c
root@localhost student]# ./a.out
Enter number of thread(upto 50) : 3
Thread....2626696960
random no obtained=33
 in thread 2626696960
 prime no list:
2thread....2618304256
random no obtained=36
in thread 2618304256
prime no list:
 2 3 5 7 11 13 17 19 23 29 31
no of prime no in the limit is 11
thread....2609911552
random no obtained=27
 in thread 2609911552
prime no list:
 3 5 7 11 13 17 19 23
No of prime no in the limit are 9
3 5 7 11 13 17 19 23 29 31
```

Experiment No- 4(b) PROCESS CREATION

Aim: Rewrite Experiment-4(a) such that the processes instead of threads are created and the number of child processes created is fixed as two. The program should make use of kernel timer to measure and print the real time, processor time user space time and kernel space for each process

DESCRIPTION:

Process is a program in execution. Process is a passive entity with a program counter specifying the next instruction to execute and a set of associated resources. Each process is represented in the operating system by a **process control block (PCB)** also called a **task control block.** It contains information such as process state, program counter, CPU registers.

System calls:

It provides the interface between a process and the operating system. It is a routine built in the kernel to perform function that requires communication with the system's hardware. All activities related to file handling; process and memory management and maintenance of user and system information are handled by the kernel using these system calls.

Kernel:

A Kernel timer is a data structure that instructs the kernel to execute a user-defined function with a user-defined argument at a user-defined time.

In computing, the kernel is the main component of most computer operating systems; it is a bridge between applications and the actual data processing done at the hardware level. The kernel's responsibilities include managing the system's resources (the communication between hardware and software components).

This program should use kernel timer to measure the following.

1) Real time.

2) User space time.

3) Kernel space time.

4) Processor time.

For both of the child processes as well as for parent process i.e., main ().

- **Processor time:** The amount of time a process takes to run, given that it has exclusive and uninterrupted use of the CPU. Note that in a modern computer, this would be very unusual, and so the processor time calculation for most processes involves adding up all the small amounts of time the CPU actually spends on the process. Some systems break processor time down into user time and system time.

- **Real time**: Real time is the amount of time spent between process creation upto the end of process execution is considered.
- **User space time:** User space is that portion of system memory in which user processes run. The amount of time spent by a process in user mode from the time of creation to its process termination is called user space time.
- **Kernel space time:** The kernel is a program that constitutes the central core of a computer operating system. It is not a process, but rather a controller of processes, and it has complete control over everything that occurs on the system. This includes managing individual user processes within user space and preventing them from interfering with each other.

Keywords, Functions & Header Files used:
1) Times () - get process times

The times () function stores the current process times in the structtms that buf points to. The structtms is as defined in <sys/times.h>:

Structtms

{

clock_ttms_utime; /* user time */

clock_ttms_stime; /* system time */

clock_ttms_cutime; /* user time of children */

clock_ttms_cstime; /* system time of children */

};

The tms_utime field contains the CPU time spent executing instructions of the calling process. The tms_stime field contains the CPU time spent in the system while executing tasks on behalf of the calling process.The tms_cutime field contains the sum of the tms_utime and tms_cutime values for all waited-for terminated children.The tms_cstime field contains the sum of the tms_stime and tms_cstime values for all waited-for terminated children.

Times for terminated children (and their descendants) are added in at the moment wait (2) or waitpid (2) returns their process ID. In particular, times of grandchildren that the children did not wait for are never seen. All times reported are in clock ticks.

Return Value: The function times returns the number of clock ticks that have elapsed since an arbitrary point in the past. For Linux this point is the moment the system was booted. This return

value may overflow the possible range of type clock_t. On error, (clock_t) -1 is returned, and errno is set appropriately.

2) Wait ():- wait for process termination.

The wait function suspends execution of the current process until a child has exited, or until a signal is delivered whose action is to terminate the current process or to call a signal handling function. If a child has already exited by the time of the call (a so-called "zombie" process), the function returns immediately. Any system resources used by the child are freed.

3) Fork ():- create a child process.

Fork creates a child process that differs from the parent process only in its PID and PPID, and in the fact that resource utilizations are set to 0. File locks and pending signals are not inherited. Under Linux, fork is implemented using copy-on-write pages, so the only penalty incurred by fork is the time and memory required to duplicate the parent's page tables, and to create a unique task structure for the child.

Return Value: On success, the PID of the child process is returned in the parent's thread of execution, and a 0 is returned in the child's thread of execution. On failure, a -1 will be returned in the parent's context, no child process will be created, and errno will be set appropriately.

4) Exit ():- cause normal program termination.

The exit () function causes normal program termination and the value of status & 0377 is returned to the parent. All functions registered with at exit () and on exit () are called in the reverse order of their registration, and all open streams are flushed and closed. Files created by tmpfile () are removed.

The C standard specifies two defines EXIT_SUCCESS and EXIT_FAILURE that may be passed toexit () to indicate successful or unsuccessful termination, respectively.

Return Value: The exit () function does not return.

5) Unistd.h: This header file is used for system call wrapper functions such asfork (), pipe () and I/Oprimitives such as read, write, close, etc.

PROGRAM CODE:

```
#include <stdio.h>

#include <stdlib.h>
```

```
#include <unistd.h>

#include <sys/times.h>

child_process(int process_no)

{

        int i, n, prime, TotPrimeNo=0;

        struct tms chpr_t1, chpr_t2;

        long rchprt1, rchprt2;

        int Rand;

        rchprt1 = times(&chpr_t1); //start time of child process

        printf("\n Processor time for child process (%d) = %d", (process_no +1), rchprt1);

        printf("\n Created child process %d ", process_no+1);

        srand(process_no);

        Rand = random() % 10000;

        printf(" \n Random number = %d\t", Rand );

        // total prime no generation

        for(n=0; n<Rand; n++)

        {

         prime = 1; // initialize

        for(i=2; i<n; i++)

        {

                        if(n % i == 0)

                        {

                        prime = 0;

                        break;

                        }

                }

                if(prime)

                {
```

```
                }
        }
        printf("\nTotal Prime Nos=%d\n", TotPrimeNo);
        rchprt2 = times(&chpr_t2); // end time of child process
        printf("---------- The real time, user space and kernel space for chid process are: ----------");
        printf("\n Real time for child process (%d) = %d", (process_no +1),
                                                (rchprt2- rchprt1));

         printf("\n User space time for child process(%d) = %d", (process_no +1),
                                                (chpr_t2.tms_cutime -chpr_t1.tms_cutime));
        printf("\n kernel space time for child process(%d) = %d\n\n", (process_no +1),
                                        (chpr_t2.tms_cstime – chpr_t1.tms_cstime));
        exit(0);
}

void CreateProcess()
{
        int i;
        for (i=0; i<2;i++)
        {
                if(fork( ) == 0)
                child_process(i);
        }
        for (i=0; i<2;i++)
        {
                wait( );
        }
}
int main ()
```

```
{
        long p_mt1, p_mt2;
        struct tms p_mst1, p_mst2
        p_mt1 = times(&p_mst1);   //start time of a parent process
        CreateProcess();
         p_mt2 = times(&p_mst2); // end time of a parent process
        printf("---------- The real time, user space and kernel space for parent process are: -------
/n");
        printf("\n Real time for parent process = %d", p_mt2-p_mt1);
        printf("\n User space time for parent process = %d",(p_mst2.tms_utime -
                                                                p_mst1.tms_utime));
        printf("\n Kernel space time for parent process = %d\n\n",(p_mst2.tms_stime -
                                                                p_mst1.tms_stime));
}
```

Output

```
[student@localhost ~]$ gcc -lpthread pp2.c
[student@localhost ~]$ ./a.out
Process Id.....4307
Random Number : 33
no of prime number 11
real time:81
cpu time:-998520
user time:-998520
kernal time:-998530
parentProcess Id.....4308
Random Number : 33
no of prime number 11
real time:62
```

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lab Manual

user time:-998520

kernal time:-998530

Experiment No-5 Producer and Consumer Problem

Aim: Design, develop and implement a process with a producer thread and a consumer thread which make use of bounded buffer (size can be prefixed at a suitable value) for communication. Use any suitable synchronization construct.

DESCRIPTION:

The "Bounded-Buffer (Producer-Consumer) problem" is to create producer thread and consumer thread allows process to communicate and to synchronize their actions with the use of a bounded buffer. The bounded-buffer problem is a paradigm for cooperating processes, where the p*roducer* process produces information that is consumed by a *consumer* process.

The producer and the consumer, who share a common, fixed-size buffer. The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time the consumer is consuming the data (i.e., removing it from the buffer) one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

The key idea behind synchronization technique is to maintain data consistency, orderly execution of cooperating processes and preventing race condition.

Statement of the Problem:

The project entitled "Inter process Communication" has been designed using bounded-buffer problem to provide better paradigm for cooperating processes compare to other problems.

Producer-consumer problem (also known as the bounded-buffer problem) is a classical example of a multiprocessor synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer. The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time the consumer is consuming the data (i.e., removing it from the buffer) one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

DESIGN:

This program can be divided mainly into three parts Producer, Consumer, Bounded-Buffer.

Producer: Producer corresponds to produce the item and stores in the buffer. After storing the item in the buffer, it has to increment the counter also it has to notice the consumer to consume the item.

Consumer consumes the item: Consumer corresponds to consume the item available in the buffer. After consuming the item in the buffer, it has to decrement the counter also it has to notice the consumer to consume the item.

Bounded-Buffer: It works as the interface between the Producer and the Consumer. It allows the producer to store the item in the buffer (shared space), also allows the consumer to consume the item from the buffer (shared space).

Producer-Consumer Problem:
Producer-consumer problem (also known as the bounded-buffer problem) is a classical example of a multiprocessor synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer. The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time the consumer is consuming the data (i.e., removing it from the buffer) one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of inter-process communication, typically using semaphores. An inadequate solution could result in a deadlock where both processes are waiting to be awakened. The problem can also be generalized to have multiple producers and consumers.

Producer-Consumer problem is a paradigm for cooperating processes; *producer* process produces information that *consumer* process consumes the information.

- Unbounded-buffer places no practical limit on the size of the buffer.
- Bounded-buffer assumes that there is a fixed buffer size.

FLOW DIAGRAM:

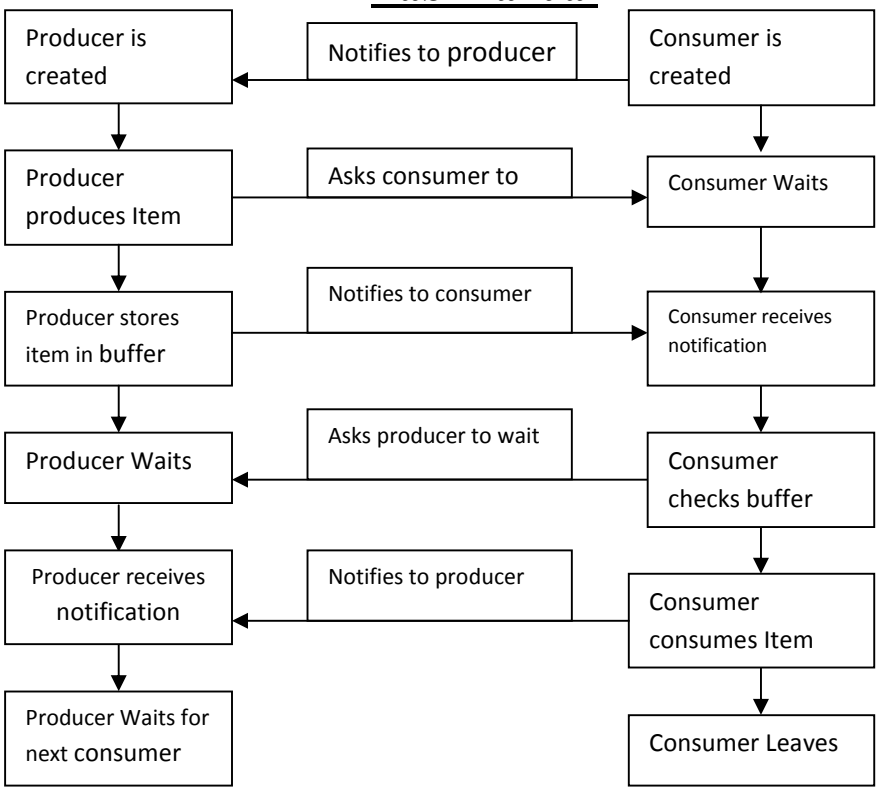


Figure 1.1: Flow diagram of Producer Consumer Problem

IMPLEMENTATION:

Algorithm

Producer process:

```
//item nextProduced;
while (buffer_size = =max_size) {
        ; /* do nothing */
                wait( );
                    }
Add item into buffer;
notify( );
```

Consumer process:

```
//item to be Consumed;
while (buffer_size = = 0) {
        ; /* do nothing */
        wait( );
    }
Remove item from buffer;
notify( );
```

PROGRAM CODE:

```
/* buffer.h */
typedef int buffer_item;
#define BUFFER_SIZE 5

/* main.c */

#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include "buffer.h"
#define RAND_DIVISOR 100000000
#define TRUE 1

/* The mutex lock */
pthread_mutex_t mutex;
/* the semaphores */
sem_t full, empty;
/* the buffer */
buffer_item buffer[BUFFER_SIZE];
/* buffer counter */
int counter;

pthread_t tid;       //Thread ID
pthread_attr_t attr; //Set of thread attributes

void *producer(void *param); /* the producer thread */
void *consumer(void *param); /* the consumer thread */

void initializeData() {
  /* Create the mutex lock */
  pthread_mutex_init(&mutex, NULL);

  /* Create the full semaphore and initialize to 0 */
  sem_init(&full, 0, 0);

  /* Create the empty semaphore and initialize to BUFFER_SIZE */
  sem_init(&empty, 0, BUFFER_SIZE);

  /* Get the default attributes */
  pthread_attr_init(&attr);

  /* init buffer */
  counter = 0;
}

/* Producer Thread */
void *producer(void *param) {
  buffer_item item;

          while(TRUE) {
```

```
                int rNum = rand() / RAND_DIVISOR;
                sleep(rNum);

                 /* generate a random number */
                item = rand();

                 /* acquire the empty lock */
                sem_wait(&empty);
                /* acquire the mutex lock */
                pthread_mutex_lock(&mutex);

                if(insert_item(item)) {
                                        fprintf(stderr, " Producer report error condition\n");
                                        }
                else {
                                printf("producer produced %d\n", item);
                      }
                /* release the mutex lock */
                pthread_mutex_unlock(&mutex);
                /* signal full */
                sem_post(&full);
                }
        }

/* Consumer Thread */
void *consumer(void *param) {
  buffer_item item;

        while(TRUE) {
                /* sleep for a random period of time */
                int rNum = rand() / RAND_DIVISOR;
                sleep(rNum);

                /* aquire the full lock */
                sem_wait(&full);
                /* aquire the mutex lock */
                pthread_mutex_lock(&mutex);
                if(remove_item(&item)) {
                                        fprintf(stderr, "Consumer report error condition\n");
                                        }
               else {
                     printf("consumer consumed %d\n", item);
                     }
                 /* release the mutex lock */
                pthread_mutex_unlock(&mutex);
                /* signal empty */
                sem_post(&empty);
          }
}

/* Add an item to the buffer */
int insert_item(buffer_item item) {
```

```
        /* When the buffer is not full add the item
        and increment the counter*/
        if(counter < BUFFER_SIZE) {
                        buffer[counter] = item;
                        counter++;
                        return 0;
                }
        else { /* Error the buffer is full */
                return -1;
              }
       }

/* Remove an item from the buffer */
int remove_item(buffer_item *item) {
        /* When the buffer is not empty remove the item
        and decrement the counter */
                if(counter > 0) {
                        *item = buffer[(counter-1)];
                        counter--;
                        return 0;
                }
        else { /* Error program code


        /* buffer.h */
        typedef int buffer_item;
        #define BUFFER_SIZE 5

        /* main.c */

#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include "buffer.h"

#define RAND_DIVISOR 100000000
#define TRUE 1


int main(int argc, char *argv[]) {
  /* Loop counter */
  int i;

  /* Verify the correct number of arguments were passed in */
  if(argc != 4) {
    fprintf(stderr, "USAGE:./main.out <INT> <INT> <INT>\n");
  }

  int mainSleepTime = atoi(argv[1]); /* Time in seconds for main to sleep */
  int numProd = atoi(argv[2]); /* Number of producer threads */
```

```
  /* Initialize the app */
  initializeData();

  /* Create the producer threads */
  for(i = 0; i < numProd; i++) {
    /* Create the thread */
    pthread_create(&tid,&attr,producer,NULL);
   }

  /* Create the consumer threads */
  for(i = 0; i < numCons; i++) {
    /* Create the thread */
    pthread_create(&tid,&attr,consumer,NULL);
  }

  /* Sleep for the specified amount of time in milliseconds */
  sleep(mainSleepTime);

  /* Exit the program */
  printf("Exit the program\n");
  exit(0);
}

/* The mutex lock */
pthread_mutex_t mutex;

/* the semaphores */
sem_t full, empty;

/* the buffer */
buffer_item buffer[BUFFER_SIZE];

/* buffer counter */
int counter;

pthread_t tid;       //Thread ID
pthread_attr_t attr; //Set of thread attributes

void *producer(void *param); /* the producer thread */
void *consumer(void *param); /* the consumer thread */

void initializeData() {

  /* Create the mutex lock */
  pthread_mutex_init(&mutex, NULL);

  /* Create the full semaphore and initialize to 0 */
  sem_init(&full, 0, 0);

  /* Create the empty semaphore and initialize to BUFFER_SIZE */
  sem_init(&empty, 0, BUFFER_SIZE);
```

```
  /* Get the default attributes */
  pthread_attr_init(&attr);

int main(int argc, char *argv[]) {
  /* Loop counter */
  int i;

  /* Verify the correct number of arguments were passed in */
  if(argc != 4) {
    fprintf(stderr, "USAGE:./main.out <INT> <INT> <INT>\n");
  }

  int mainSleepTime = atoi(argv[1]); /* Time in seconds for main to sleep */
  int numProd = atoi(argv[2]); /* Number of producer threads */
  int numCons = atoi(argv[3]); /* Number of consumer threads */

  /* Initialize the app */
  initializeData();

  /* Create the producer threads */
  for(i = 0; i < numProd; i++) {
    /* Create the thread */
    pthread_create(&tid,&attr,producer,NULL);
   }

  /* Create the consumer threads */
  for(i = 0; i < numCons; i++) {
    /* Create the thread */
    pthread_create(&tid,&attr,consumer,NULL);
  }

  /* Sleep for the specified amount of time in milliseconds */
  sleep(mainSleepTime);

  /* Exit the program */
  printf("Exit the program\n");
  exit(0);
}nit buffer */
  counter = 0;
}

/* Producer Thread */
void *producer(void *param) {
  buffer_item item;

  while(TRUE) {
    /* sleep for a random period of time */
    int rNum = rand() / RAND_DIVISOR;
    sleep(rNum);

    /* generate a random number */
```

```
    /* acquire the empty lock */
    sem_wait(&empty);
    /* acquire the mutex lock */
    pthread_mutex_lock(&mutex);

    if(insert_item(item)) {
      fprintf(stderr, " Producer report error condition\n");
    }
    else {
      printf("producer produced %d\n", item);
    }
    /* release the mutex lock */
    pthread_mutex_unlock(&mutex);
    /* signal full */
    sem_post(&full);
  }
}

/* Consumer Thread */
void *consumer(void *param) {
  buffer_item item;

  while(TRUE) {
    /* sleep for a random period of time */
    int rNum = rand() / RAND_DIVISOR;
    sleep(rNum);

    /* aquire the full lock */
    sem_wait(&full);
    /* aquire the mutex lock */
    pthread_mutex_lock(&mutex);
    if(remove_item(&item)) {
      fprintf(stderr, "Consumer report error condition\n");
    }
    else {
      printf("consumer consumed %d\n", item);
    }
    /* release the mutex lock */
    pthread_mutex_unlock(&mutex);
    /* signal empty */
    sem_post(&empty);
  }
}

/* Add an item to the buffer */
int insert_item(buffer_item item) {
  /* When the buffer is not full add the item
    and increment the counter*/
  if(counter < BUFFER_SIZE) {
    buffer[counter] = item;
    counter++;
```

```
    return 0;
  }
  else { /* Error the buffer is full */
    return -1;
  }
}

/* Remove an item from the buffer */
int remove_item(buffer_item *item) {
  /* When the buffer is not empty remove the item
    and decrement the counter */
  if(counter > 0) {
    *item = buffer[(counter-1)];
    counter--;
    return 0;
  }
  else { /* Error buffer empty */
    return -1;
  }
}

int main(int argc, char *argv[]) {
  /* Loop counter */
  int i;

  /* Verify the correct number of arguments were passed in */
  if(argc != 4) {
    fprintf(stderr, "USAGE:./main.out <INT> <INT> <INT>\n");
  }

  int mainSleepTime = atoi(argv[1]); /* Time in seconds for main to sleep */
  int numProd = atoi(argv[2]); /* Number of producer threads */
  int numCons = atoi(argv[3]); /* Number of consumer threads */

  /* Initialize the app */
  initializeData();

  /* Create the producer threads */
  for(i = 0; i < numProd; i++) {
    /* Create the thread */
    pthread_create(&tid,&attr,producer,NULL);
   }

  /* Create the consumer threads */
  for(i = 0; i < numCons; i++) {
    /* Create the thread */
    pthread_create(&tid,&attr,consumer,NULL);
  }

  /* Sleep for the specified amount of time in milliseconds */
  sleep(mainSleepTime);
```

```
  /* Exit the program */
  printf("Exit the program\n");
  exit(0);
}
```

OUTPUT :

```
File  Edit  View  Search  Terminal  Help
[student@localhost ~]$ gcc -lpthread pro3.c
[student@localhost ~]$ ./a.out 10 10 10
producer produced 35005211
consumer consumed 35005211
producer produced 1726956429
consumer consumed 1726956429
producer produced 278722862
consumer consumed 278722862
producer produced 468703135
producer produced 1801979802
producer produced 635723058
producer produced 1125898167
consumer consumed 1125898167
Exit the program
[student@localhost ~]$
```

Experiment No- 6 Remote Procedure Call
Aim: Design, develop, and execute a program to demonstrate the use of RPC.

DESCRIPTION:
Procedure Call (RPC) defines a powerful technology for creating distributed client/server programs. The RPC run-time stubs and libraries manage most of the processes relating to network protocols and communication. This enables you to focus on the details of the application rather than the details of the network.

Examples: File service, Database service, and Authentication service.

Need of RPC: The client needs an easy way to call the procedures of the server to get some services.RPC enables clients to communicate with servers by calling procedures in a similar way to the conventional use of procedure calls in high-level languages. RPC is modeled on the local procedure call, but the called procedure is executed in a different process and usually a different computer.

RMI Architecture:

RMI system allows an object running in one JVM to invoke methods on an object in another JVM. RMI can communicate only between programs written in Java.

RMI applications often consist of two applications: A sever and a client

The application is divided into objects. A server creates remote objects and registers these objects with a simple bootstrap name server called a registry (rmiregistry). The client gets a handle by looking up the remote object by its name in the server(s) registry and invokes methods on it.

Remote object :

A remote object is one whose methods can be invoked from another JVM practically on a different machine. In situations where you do not have a network, or a client, or a server available to test programs, you can invoke methods on a remote object from another JVM on the same machine. A remote object must implement at least one interface that extends the java.rmi.Remote interface.

Remote Interface :

A remote interface is an interface, which declares a set of methods that may be invoked remotely (from a different JVM). All interactions with the remote object will be performed through this interface. A remote interface must directly or indirectly extend the java.rmi.Remote interface. Each method declaration in a remote interface must include java.rmi.RemoteException or one of its super classes in its throws clause.

The Remote Procedure Call (RPC) message protocol consists of two distinct structures: the call message and the reply message.

RPC Call Message: Each remote procedure call message contains the following unsigned integer fields to uniquely identify the remote procedure:

- ✓ Program number
- ✓ Program version number
- ✓ Procedure number

The body of an RPC call message takes the following form:

```
struct call_body {

unsigned int rpcvers;

unsigned int prog;

unsigned int vers;

unsigned int proc;
```

opaque_auth verf;

1 parameter 2 parameter.

};

RPC Reply Message: The RPC protocol for a reply message varies depending on whether the call message is accepted or rejected by the network server. The reply message to a request contains information to distinguish the following conditions:

- RPC executed the call message successfully.
- The remote implementation of RPC is not protocol version 2. The lowest and highest supported RPC version numbers are returned.
- The remote program is not available on the remote system.
- The remote program does not support the requested version number. The lowest and highest supported remote program version numbers are returned.
- The requested procedure number does not exist. This is usually a caller-side protocol or programming error.

The RPC reply message takes the following form:

enum reply_stat stat {

MSG_ACCEPTED = 0,

MSG_DENIED= 1

};

Function used:

- **XmlRpcClientConfigImpl():**
  Default implementation of a clients request configuration.
- **xmlRpcServer():**
  This is the embedded XML-RPC server, which is called to execute the clients requests Obviously, this is an extremely fast transport. However, its main use is for debugging and development.

PROGRAM CODE:

//**SERVER PROGRAM**//

import org.apache.xmlrpc.server.PropertyHandlerMapping;

```
import org.apache.xmlrpc.webserver.WebServer;

public class RpcServer {

        public Integer sum(int x, int y) {

                        System.out.println("Finding sum for " + x + " and " + y);

                        return new Integer(x + y);

                        }

        public static void main(String[] args) {

                        try {

                                System.out.println("Attempting to start XML-RPC Server...");

                                WebServer server = new WebServer(8080);

                                PropertyHandlerMapping phm = new PropertyHandlerMapping();

                                phm.addHandler("math", RpcServer.class);

                                server.getXmlRpcServer().setHandlerMapping(phm);

                                server.start();

                                System.out.println("Started successfully.");

                                System.out.println("Accepting requests. (Halt program to stop.)");

                                }

                catch (Exception exception) {

                                System.err.println("JavaServer: " + exception);

                        }

                }

        }
```

//CLIENT PROGRAM**//**

```
import java.net.URL;
```

```
import java.util.Vector;

import org.apache.xmlrpc.client.XmlRpcClient;

import org.apache.xmlrpc.client.XmlRpcClientConfigImpl;

public class RpcClient {

        public static void main(String[] args) throws Exception {

                XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();

                String serverAddress = "localhost:8080";

                int val1 = 0;

                int val2 = 0;

                if (args.length > 0) {

                                serverAddress = args[0];

                                }

                if (args.length >= 2) {

                                val1 = Integer.parseInt(args[1]);

                                val2 = Integer.parseInt(args[2]);

                                }

                config.setServerURL(new URL("http://" + serverAddress + "/RPC2"));

                XmlRpcClient server = new XmlRpcClient();

                server.setConfig(config);

                Vector<Integer> params = new Vector<Integer>();

                params.addElement(new Integer(val1));

                params.addElement(new Integer(val2));

                Object result = server.execute("math.sum", params);

                int sum = ((Integer) result).intValue();

                System.out.println("The sum is: " + sum);
```

```
        }
}
```

Output

Server:

Attempting to start XML-RPC Server...

Started successfully.

Accepting requests. (Halt program to stop.)

Finding sum for 17 and 13

Finding sum for 54 and 23

Client:

$ java RpcClient localhost:8080 17 13

The sum is: 30

$ java RpcClient localhost:8080 54 23

The sum is: 77

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lab Manual

Title of Course: Advanced Programming Lab
Course Code: MCSE192
L-T-P scheme: 0-0-3 **Course Credit: 2**

Objectives:
1. To learn and understand different types artificial neural network algorithm.
2. To learn MatLab for the programming of ANN.

Learning Outcomes: The students will have a detailed knowledge of the concepts of matlab. Upon the completion of Advanced algorithm course, the student will be able to:

- **Understand** and implement basic services and functionalities of the ANN using matlab.
- **Use** KohonenSelfOrganizingfeaturemaptoClusterthe vectorsusingowninitial weightsand learningrate.
- **Understand** the benefits of artificial neural network in artificial intelligence.

Course Contents:
Exercises that must be done in this course are listed below:
Exercise1: Programtogeneratea few activationfunctionthat are being used in neural network
Exercise2: Programtoclassifywith a 2-inputperceptron.
Exercise 3: Program for perceptron net for an AND function with bipolar inputs and targets.
Exercise 4: Developa MatlabprogramforOR functionwith bipolar inputsand targetsusingADALINEnetwork.
Exercise 5: Developa MatlabprogramtogenerateXOR functionforbipolar inputsand targetsusingMADALINENetwork.
Exercise 6: Developa Matlabprogramtostorethe vector(-1,-1,-1,-1)and(-1,-1,1,1)inanauto-associativenetwork.Findtheweight matrix.Test thenet with (1,1,1,1)asinput.
Exercise 7: Considera vector(1,0,1,1)to bestoredinthenet.Test a discreteHopfieldnet with error inthe1stand4thcomponents (0,0,1,0)ofthestoredvector.
Exercise 8: Developa MatlabprogramforXOR function (binaryinputandoutput) with momentumfactorusingback-propagation algorithm.
Exercise 9: DevelopMatlabprogramfordrawingfeature maps(KohonenSelfOrganizingFeature maps)in 1-Dimensionalview.
Exercise 10: UseKohonenSelfOrganizingfeaturemaptoClusterthe vectors(assumefourbinaryvectors)usingowninitial weights(to be assumed)and learningrate(tobe assumed).

Text Book:
1. S. N. Sivanandam, S. N Deepa, Introduction to Neural Networks Using Matlab 6.0, 1st edition, Tata McGraw-Hill Education, 2006

References:
1. B.Yegnanarayana, ArtificialNeuralNetworks,PrenticeHall ofIndia,1999
2. SatishKumar,NeuralNetworks–AClassroomApproach,Tata McGraw-Hill,2003
3. S.Haykin,NeuralNetworks–AComprehensiveFoundation,PrenticeHall,1998
4. C.M.Bishop,PatternRecognitionand MachineLearning,Springer,2006

Recommended Systems/Software Requirements:
1. Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
2. Matlab software in Windows XP .

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lab Manual

Exercise1: Generate a few activationfunctionthat are being used in neural network

Description:

Activation Function :

A function used to transform the activation level of a unit (neuron) into an output signal. Typically, activation functions have a "squashing" effect. Together with the PSP function (which is applied first) this defines the unit type.Neural Networks supports a wide range of activation functions. Only a few of these are used by default; the others are available for customization.

- Identity: The activation level is passed on directly as the output. Used in a variety of network types, including linear networks, and the output layer of radial basis function networks.
- Logistic: This is an S-shaped (sigmoid) curve, with output in the range (0,1).
- Hyperbolic: The hyperbolic tangent function (tanh): a sigmoid curve, like the logistic function, except that output lies in the range (-1,+1). Often performs better than the logistic function because of its symmetry. Ideal for customization of multilayer perceptrons, particularly the hidden layers.
- Exponential: The negative exponential function. Ideal for use with radial units. The combination of radial synaptic function and negative exponential activation function produces units that model a Gaussian (bell-shaped) function centred at the weight vector.
- Softmax: Exponential function, with results normalized so that the sum of activations across the layer is 1.0. Can be used in the output layer of multilayer perceptrons for classification problems, so that the outputs can be interpreted as probabilities of class membership (Bishop, 1995; Bridle, 1990).
- Unit sum: Normalizes the outputs to sum to 1.0. Used in PNNs to allow the outputs to be interpreted as probabilities.
- Square root: Used to transform the squared distance activation in an SOFM network or Cluster network to the actual distance as an output.
- Sine: Possibly useful if recognizing radially-distributed data; not used by default.
- Ramp: A piece-wise linear version of the sigmoid function. Relatively poor training performance, but fast execution.
- Step: Outputs either 1.0 or 0.0, depending on whether the Synaptic value is positive or negative. Can be used to model simple networks such as perceptrons.

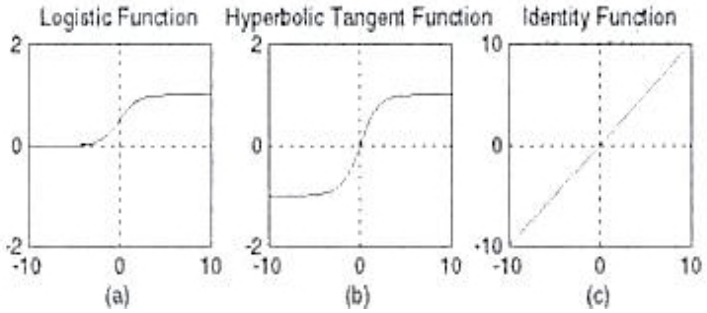
Aim: Programtogenerate a few activationfunctionthat are being used in neural network

Program

```
% Illustration of various activation functions used in NN's
x = -10:0.1:10;
tmp = exp(-x);
y1 = 1./(1+tmp);
y2 = (1-tmp)./(1+tmp);
y3 = x;
subplot(231); plot(x, y1); grid on;
axis([min(x) max(x) -2 2]);
title('Logistic Function');
xlabel('(a)');
axis('square');
subplot(232); plot(x, y2); grid on;
axis([min(x) max(x) -2 2]);
```

```
xlabel('(b)');
axis('square');
subplot(233); plot(x, y3); grid on;
axis([min(x) max(x) min(x) max(x)]);
title('Identity Function');
xlabel('(c)');
axis('square');
```

Output:



Exercise2: Programtoclassifywith a 2-inputperceptron.
Description:

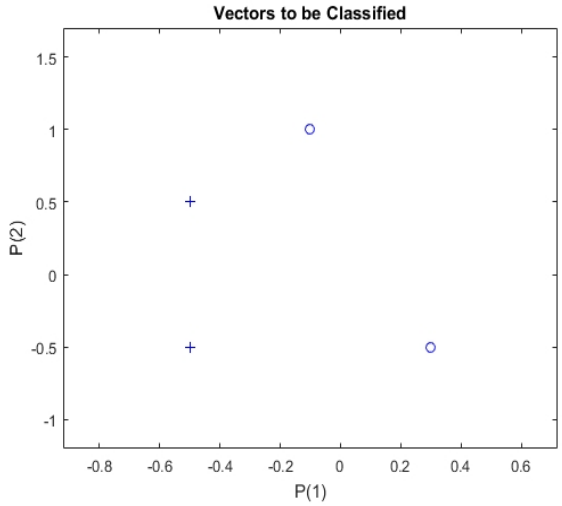
A 2-input hard limit neuron is trained to classify 5 input vectors into two categories.

Each of the five column vectors in X defines a 2-element input vectors and a row vector T defines the vector's target categories. We can plot these vectors with PLOTPV.

Program(A):

```
X = [ -0.5 -0.5 +0.3 -0.1;
     -0.5 +0.5 -0.5 +1.0];
T = [1 1 0 0];
plotpv(X,T);
```

output:



The perceptron must properly classify the 5 input vectors in X into the two categories defined by T. Perceptrons have HARDLIM neurons. These neurons are capable of separating an input space with a straight line into two categories (0 and 1).

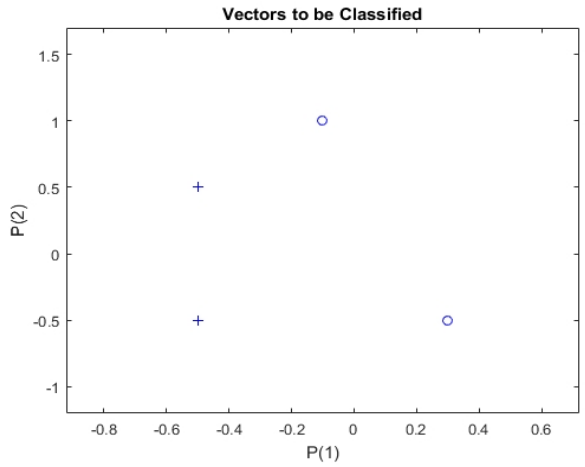
Here PERCEPTRON creates a new neural network with a single neuron. The network is then configured to the data, so we can examine its initial weight and bias values. (Normally the configuration step can be skipped as it is automatically done by ADAPT or TRAIN.)

```
net = perceptron;
net = configure(net,X,T);
```

The input vectors are replotted with the neuron's initial attempt at classification.

The initial weights are set to zero, so any input gives the same output and the classification line does not even appear on the plot. Fear not... we are going to train it!

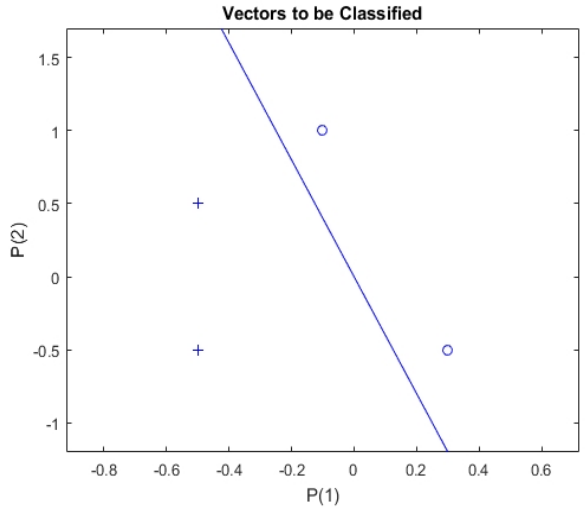
```
plotpv(X,T);
plotpc(net.IW{1},net.b{1});
```



Here the input and target data are converted to sequential data (cell array where each column indicates a timestep) and copied three times to form the series XX and TT.

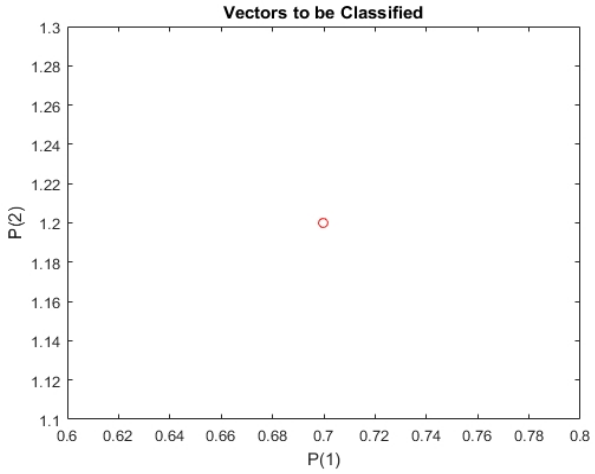
ADAPT updates the network for each timestep in the series and returns a new network object that performs as a better classifier.

```
XX = repmat(con2seq(X),1,3);
TT = repmat(con2seq(T),1,3);
net = adapt(net,XX,TT);
plotpc(net.IW{1},net.b{1});
```



Now SIM is used to classify any other input vector, like [0.7; 1.2]. A plot of this new point with the original training set shows how the network performs. To distinguish it from the training set, color it red.

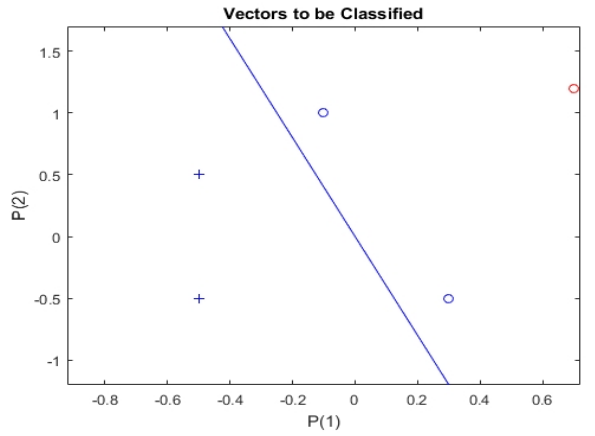
```
x = [0.7; 1.2];
y = net(x);
plotpv(x,y);
point = findobj(gca,'type','line');
point.Color = 'red';
```



Turn on "hold" so the previous plot is not erased and plot the training set and the classification line.

The perceptron correctly classified our new point (in red) as category "zero" (represented by a circle) and not a "one" (represented by a plus).

```
hold on;
plotpv(X,T);
plotpc(net.IW{1},net.b{1});
hold off;
```



Exercise 3: Perceptron net for an AND function with bipolar inputs and targets.

Description:

A perceptron is an artificial neuron using the Heaviside step function as the activation function. The perceptron algorithm is also termed the single-layer perceptron, to distinguish it from a multilayer perceptron, which is a misnomer for a more complicated neural network. As a linear classifier, the single-layer perceptron is the simplest feed forward neural network.

Aim: Program for perceptron net for an AND function with bipolar inputs and targets.

The truth table for the AND function is given as

X_1	X_2	Y

– 1	1	– 1
1	– 1	– 1
1	1	1

The MATLAB program for the above table is given as follows.

Program

```
%Perceptron for AND funtion
clear;
clc;
x=[1 1 -1 -1;1 -1 1 -1];
t=[1 -1 -1 -1];
w=[0 0];
b=0;
alpha=input('Enter Learning rate=');
theta=input('Enter Threshold value=');
con=1;
epoch=0;
while con
   con=0;
   for i=1:4
      yin=b+x(1,i)*w(1)+x(2,i)*w(2);
      if yin>theta
         y=1;
      end
      if yin <=theta & yin>=-theta
         y=0;
      end
      if yin<-theta
         y=-1;
      end
      if y-t(i)
         con=1;
         for j=1:2
            w(j)=w(j)+alpha*t(i)*x(j,i);
         end
         b=b+alpha*t(i);
      end
   end
   epoch=epoch+1;
end
disp('Perceptron for AND funtion');
disp(' Final Weight matrix');
disp(w);
disp('Final Bias');
disp(b);
```

Output

```
Enter Learning rate=1
Enter Threshold value=0.5
Perceptron for AND funtion
Final Weight matrix
     1     1
Final Bias
        -1
```

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lab Manual

Exercise 4: OR functionwith bipolar inputsand targetsusingADALINEnetwork.

Description:
ADALINE (**Adaptive Lineardi Neuron** or later **Adaptive Linear Element**) is an early single-layer artificial neural network and the name of the physical device that implemented this network.

Aim:**Write aMatlabprogramforOR functionwith bipolar inputsand targetsusingADALINEnetwork.**
The truth table for the OR function with bipolar inputs and targets is given as,

X_1	X_2	Y
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

Program:

```
clear all;
clc;
disp('Adaline network for OR function Bipolar inputs and targets');
%input pattern
x1=[1 1 -1 -1];
x2=[1 -1 1 -1];
%bias input
x3=[1 1 1 1];
%target vector
t=[1 1 1 -1];
%initial weights and bias
w1=0.1;w2=0.1;b=0.1;
%initialize learning rate
alpha=0.1;
%error convergence
e=2;
%change in weights and bias
delw1=0;delw2=0;delb=0;
epoch=0;
while(e>1.018)
   epoch=epoch+1;
   e=0;
   for i=1:4
      nety(i)=w1*x1(i)+w2*x2(i)+b;
      %net input calculated and target
      nt=[nety(i) t(i)];
      delw1=alpha*(t(i)-nety(i))*x1(i);
      delw2=alpha*(t(i)-nety(i))*x2(i);
      delb=alpha*(t(i)-nety(i))*x3(i);
      %weight changes
      wc=[delw1 delw2 delb]
      %updating of weights
      w1=w1+delw1;
      w2=w2+delw2;
      b=b+delb;
      %new weights
      w=[w1 w2 b]
      %input pattern
```

```
    %printring the results obtained
    pnt=[x nt wc w]
end
for i=1:4
        nety(i)=w1*x1(i)+w2*x2(i)+b;
        e=e+(t(i)-nety(i))^2;
end
  end
```

Output

	X1	X2	b	Netin	Target	delw1	delw2	delb	w1(n)	w2(n)	b(n)
Epoch 1	1.0000	1.0000	1.0000	0.3000	1.0000	0.0700	0.0700	0.0700	0.1700	0.1700	0.1700
	1.0000	–1.0000	1.0000	0.1700	1.0000	0.0830	–0.0830	0.0830	0.2530	0.0870	0.2530
	–1.0000	1.0000	1.0000	0.0870	1.0000	–0.0913	0.0913	0.0913	0.1617	0.1783	0.3443
	–1.0000	–1.0000	1.0000	0.0043	–1.0000	0.1004	0.1004	–0.1004	0.2621	0.2787	0.2439
Epoch 2	1.0000	1.0000	1.0000	0.7847	1.0000	0.0215	0.0215	0.0215	0.2837	0.3003	0.2654
	1.0000	–1.0000	1.0000	0.2488	1.0000	0.0751	–0.0751	0.0751	0.3588	0.2251	0.3405
	–1.0000	1.0000	1.0000	0.2069	1.0000	–0.0793	0.0793	0.0793	0.2795	0.3044	0.4198
	–1.0000	–1.0000	1.0000	–0.1641	–1.0000	0.0836	0.0836	–0.0836	0.3631	0.3880	0.3362
Epoch 3	1.0000	1.0000	1.0000	1.0873	1.0000	-0.0087	–0.0087	–0.0087	0.3543	0.3793	0.3275
	1.0000	–1.0000	1.0000	0.3025	1.0000	0.0697	–0.0697	0.0697	0.4241	0.3096	0.3973
	–1.0000	1.0000	1.0000	0.2827	1.0000	–0.0717	0.0717	0.0717	0.3523	0.3813	0.4690
	–1.0000	–1.0000	1.0000	–0.2647	–1.0000	0.0735	0.0735	–0.0735	0.4259	0.4548	0.3954
Epoch 4	1.0000	1.0000	1.0000	1.2761	1.0000	–0.0276	–0.0276	–0.0276	0.3983	0.4272	0.3678
	1.0000	–1.0000	1.0000	0.3389	1.0000	0.0661	–0.0661	0.0661	0.4644	0.3611	0.4339
	–1.0000	1.0000	1.0000	0.3307	1.0000	–0.0669	0.0669	0.0669	0.3974	0.4280	0.5009
	–1.0000	–1.0000	1.0000	–0.3246	–1.0000	0.0675	0.0675	–0.0675	0.4650	0.4956	0.4333
Epoch 5	1.0000	1.0000	1.0000	1.3939	1.0000	–0.0394	–0.0394	–0.0394	0.4256	0.4562	0.3939
	1.0000	–1.0000	1.0000	0.3634	1.0000	0.0637	–0.0637	0.0637	0.4893	0.3925	0.4576
	–1.0000	1.0000	1.0000	0.3609	1.0000	–0.0639	0.0639	0.0639	0.4253	0.4564	0.5215
	–1.0000	–1.0000	1.0000	–0.3603	–1.0000	0.0640	0.0640	–0.0640	0.4893	0.5204	0.4575

Exercise 5: XOR functionforbipolar inputsand targetsusingMADALINENetwork.

Description:

MADALINE (**Many ADALINE**[1]) is a three-layer (input, hidden, output), fully connected, feed-forward artificial neural network architecture for classification that uses ADALINE units in its hidden and output layers, i.e. its activation function is the sign function.[2] The three-layer network uses memistors. Three different training algorithms for MADALINE networks, which cannot be learned using backpropagation because the sign function is not differentiable, have been suggested, called Rule I, Rule II and Rule III. The first of these dates back to 1962 and cannot adapt the weights of the hidden-output connection.[3] The second training algorithm improved on Rule I and was described in 1988.[1] The third "Rule" applied to a modified network with sigmoid activations instead of signum; it was later found to be equivalent to backpropagation.[3]

The Rule II training algorithm is based on a principle called "minimal disturbance". It proceeds by looping over training examples, then for each example, it:

- finds the hidden layer unit (ADALINE classifier) with the lowest confidence in its prediction,
- tentatively flips the sign of the unit,
- accepts or rejects the change based on whether the network's error is reduced

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lab Manual

Aim: Writea Matlab programto generate XOR function for bipolar inputs and targets using MADALINENetwork.

The truth table for XOR function with bipolar inputs and targets is given as,

X_1	X_2	Y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Program

```
%Madaline for XOR funtion
clc;
clear;
%Input and Target
x=[1 1 –1 –1;1 –1 1 –1];
t=[-1 1 1 –1];
%Assume initial weight matrix and bias
w=[0.05 0.1;0.2 0.2];
b1=[0.3 0.15];
v=[0.5 0.5];
b2=0.5;
con=1;
alpha=0.5;
epoch=0;
while con
   con=0;
   for i=1:4
      for j=1:2
         zin(j)=b1(j)+x(1,i)*w(1,j)+x(2,i)*w(2,j);
         if zin(j)>=0
            z(j)=1;
         else
            z(j)=–1;
         end
      end
      yin=b2+z(1)*v(1)+z(2)*v(2);
      if yin>=0
         y=1;
      else
         y=–1;
      end
      if y~=t(i)
         con=1;
         if t(i)==1
            if abs(zin(1)) > abs(zin(2))
               k=2;
            else
               k=1;
            end
            b1(k)=b1(k)+alpha*(1-zin(k));
            w(1:2,k)=w(1:2,k)+alpha*(1-zin(k))*x(1:2,i);
         else
            for k=1:2
```

```
                w(1:2,k)=w(1:2,k)+alpha*(-1-zin(k))*x(1:2,i);
              end
            end
          end
        end
    end
    epoch=epoch+1;
end
disp('Weight matrix of hidden layer');
disp(w);
disp('Bias of hidden layer');
disp(b1);
disp('Total Epoch');
disp(epoch);
```

Output:

```
Weight matrix of hidden layer
    1.3203   –1.2922
   –1.3391    1.2859
Bias of hidden layer
   –1.0672   –1.0766
Total Epoch
     3
```

Exercise 6: Storethe vector(-1,-1,-1,-1)and(-1,-1,1,1)inanauto-associativenetwork.Findtheweight matrix.Test thenet with (1,1,1,1)asinput.

Description:

Auto-associative networks are a special subset of the hetero-associative networks(Hetero-associative networks map m input vectors x1, x2,..., xm in ndimensional space to m output vectors y1, y2,..., ym in k-dimensional space, so that xi → yi . If x˜ − xi 2 < ε then x˜ → yi . This should be achieved by the learning algorithm, but becomes very hard when the number m of vectors to be learned is too high.), in which each vector is associated with itself, i.e., yi = xi for i = 1,...,m. The function of such networks is to correct noisy input vectors.

Aim: Write a Matlabprogramtostorethe vector(-1,-1,-1,-1)and(-1,-1,1,1)inanauto-associativenetwork.Findtheweight matrix.Test thenet with (1,1,1,1)asinput.

Program

```
clc;
clear;
x=[–1 –1 –1 –1;–1 –1 1 1];
t=[1 1 1 1];
w=zeros(4,4);
for i=1:2
   w=w+x(i,1:4)'*x(i,1:4);
end
yin=t*w;
for i=1:4
   if yin(i)>0
      y(i)=1;
   else
      y(i)=–1;
```

```
end
disp('The calculated weight matrix');
disp(w);
if x(1,1:4)==y(1:4) | x(2,1:4)==y(1:4)
   disp('The vector is a Known Vector');
else
   disp('The vector is a unknown vector');
end
```

Output:

```
   The calculated weight matrix
      2    2    0    0
      2    2    0    0
      0    0    2    2
      0    0    2    2
The vector is an unknown vector.
```

Exercise 7: Considera vector(1,0,1,1)to bestoredinthenet.Test a discreteHopfieldnet with error inthe1st and4thcomponents (0,0,1,0)ofthestoredvector.

Description:

A Hopfield network is a form of recurrent artificial neural network popularized by John Hopfield in 1982, but described earlier by Little in 1974.Hopfield nets serve as content-addressable memory systems with binary threshold nodes. They are guaranteed to converge to a local minimum, but will sometimes converge to a false pattern (wrong local minimum) rather than the stored pattern (expected local minimum). Hopfield networks also provide a model for understanding human memory.

Aim: Write a MATlab program to test a discreteHopfieldnet with error inthe1st and4thcomponents (0,0,1,0)ofthestoredvector.Considera vector(1,0,1,1)to bestoredinthenet.

Program:

```
%Discrete Hopfield net
clc;
clear;
x=[1 1 1 0];
tx=[0 0 1 0];
w=(2*x'–1)*(2*x–1);
for i=1:4
   w(i,i)=0;
end
con=1;
y=[0 0 1 0];
while con
   up=[4 2 1 3];
   for i=1:4
      yin(up(i))=tx(up(i))+y*w(1:4,up(i));
      if yin(up(i))>0
         y(up(i))=1;
      end
   end
   if y==x
      disp('Convergence has been obtained');
      disp('The Converged Ouput');
      disp(y);
```

```
   end
end
```

Output

Convergence has been obtained

The Converged Ouput

1 1 1 0

Exercise 8: XOR function (binaryinputandoutput) with momentumfactorusingback-propagation algorithm.

Description:

The backward propagation of errors or backpropagation, is a common method of training artificial neural networks and used in conjunction with an optimization method such as gradient descent. The algorithm repeats a two phase cycle, propagation and weight update. When an input vector is presented to the network, it is propagated forward through the network, layer by layer, until it reaches the output layer. The output of the network is then compared to the desired output, using a loss function, and an error value is calculated for each of the neurons in the output layer. The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which roughly represents its contribution to the original output.

Backpropagation uses these error values to calculate the gradient of the loss function with respect to the weights in the network. In the second phase, this gradient is fed to the optimization method, which in turn uses it to update the weights, in an attempt to minimize the loss function.

The importance of this process is that, as the network is trained, the neurons in the intermediate layers organize themselves in such a way that the different neurons learn to recognize different characteristics of the total input space. After training, when an arbitrary input pattern is present which contains noise or is incomplete, neurons in the hidden layer of the network will respond with an active output if the new input contains a pattern that resembles a feature that the individual neurons have learned to recognize during their training.

Backpropagation requires a known, desired output for each input value in order to calculate the loss function gradient – it is therefore usually considered to be a supervised learning method; nonetheless, it is also used in some unsupervised networks such as auto-encoders. It is a generalization of the delta rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer. Backpropagation requires that the activation function used by the artificial neurons (or "nodes") be differentiable.

Aim:Writea MatlabprogramforXOR function (binaryinputandoutput) with momentumfactorusingback-propagation algorithm.

Program

```
%Back Propagation Network for Data Compression
clc;
clear;
%Get Input Pattern from file
data=open('comp.mat');
x=data.x;
t=data.t;
%Input,Hidden and Output layer definition
n=63;
m=63;
h=24;
%Initialize weights and bias
v=rand(n,h)—0.5;
v1=zeros(n,h);
```

```
b2=rand(1,m)—0.5;
w=rand(h,m)—0.5;
w1=zeros(h,m);
alpha=0.4;
mf=0.3;
con=1;
epoch=0;
while con
   e=0;
   for I=1:10
      %Feed forward
      for j=1:h
         zin(j)=b1(j);
         for i=1:n
            zin(j)=zin(j)+x(I,i)*v(i,j);
         end
         z(j)=bipsig(zin(j));
      end
      for k=1:m
         yin(k)=b2(k);
      for j=1:h
            yin(k)=yin(k)+z(j)*w(j,k);
         end
         y(k)=bipsig(yin(k));
         ty(I,k)=y(k);
      end
      %Backpropagation of Error
      for k=1:m
         delk(k)=(t(I,k)-y(k))*bipsig1(yin(k));
      end
      for j=1:h
         for k=1:m
                              delw(j,k)=alpha*delk(k)*z(j)+mf*(w(j,k)—w1(j,k));
                              delinj(j)=delk(k)*w(j,k);
         end
      end
      delb2=alpha*delk;
      for j=1:h
         delj(j)=delinj(j)*bipsig1(zin(j));
      end
      for j=1:h
         for i=1:n
            delv(i,j)=alpha*delj(j)*x(I,i)+mf*(v(i,j)–v1(i,j));
         end
      end
      delb1=alpha*delj;
      w1=w;
      v1=v;
      %Weight updation
      w=w+delw;
      b2=b2+delb2;
      v=v+delv;
      b1=b1+delb1;
      for k=1:k
     e=e+(t(I,k)—y(k))^2;
   end
```

```
   if e<0.005
      con=0;
   end
   epoch=epoch+1;
   if epoch==30
      con=0;
   end
   xl(epoch)=epoch;
   yl(epoch)=e;
end
disp('Total Epoch Performed');
disp(epoch);
disp('Error');
disp(e);
figure(1);
k=1;
for i=1:2
  for j=1:5
      charplot(x(k,:),10+(j–1)*15,30–(i–1)*15,9,7);
      k=k+1;
   end
end
title('Input Pattern for Compression');
axis([0 90 0 40]);
figure(2);
plot(xl,yl);
xlabel('Epoch Number');
ylabel('Error');
title('Conversion of Net');
%Output of Net after training
for I=1:10
   for j=1:h
      zin(j)=b1(j);
      for i=1:n
         zin(j)=zin(j)+x(I,i)*v(i,j);
      end
      z(j)=bipsig(zin(j));
   end
   for k=1:m
      yin(k)=b2(k);
      for j=1:h
         yin(k)=yin(k)+z(j)*w(j,k);
      end
      y(k)=bipsig(yin(k));
      ty(I,k)=y(k);
   end
end
for i=1:10
   for j=1:63
      if ty(i,j)>=0.8
         tx(i,j)=1;
      else if ty(i,j)<=-0.8
            tx(i,j)=–1;
         else
            tx(i,j)=0;
         end
```

```
   end
end
figure(3);
k=1;
for i=1:2
   for j=1:5
      charplot(tx(k,:),10+(j–1)*15,30-(i–1)*15,9,7);
      k=k+1;
   end
end
axis([0 90 0 40]);
title('Decompressed Pattern');
```

subfuntion used:

```
%Plot character
function charplot(x,xs,ys,row,col)
k=1;
for i=1:row
   for j=1:col
      xl(i,j)=x(k);
      k=k+1;
   end
end
for i=1:row
   for j=1:col
      if xl(i,j)==1
         plot(j+xs–1,ys–i+1,'k*');
         hold on
      else
         plot(j+xs–1,ys–i+1,'r');
         hold on
      end
   end
end
function y=bipsig(x)
y=2/(1+exp(-x))–1;
function y=bipsig1(x)
y=1/2*(1-bipsig(x))*(1+bipsig(x));
```

Output

```
   (i) Learning Rate:0.5
   Momentum Factor:0.5
   Total Epoch Performed
      30
   Error
     68.8133
```

Exercise 9: KohonenSelfOrganizingFeature mapsin 1-Dimensionalview.

Description:

A self-organizing map (SOM) or self-organising feature map (SOFM) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map, and is therefore a method to do dimensionality reduction. Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning (such as backpropagation with gradient descent), and in the sense that they use a neighbourhood function to preserve the topological properties of the input space.This makes SOMs useful

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Lab Manual

The artificial neural network introduced by the Finnish professor Teuvo Kohonen in the 1980s is sometimes called a Kohonen map or network. The Kohonen net is a computationally convenient abstraction building on work on biologically neural models from the 1970s and morphogenesis models dating back to Alan Turing in the 1950s.Like most artificial neural networks, SOMs operate in two modes: training and mapping. "Training" builds the map using input examples (a competitive process, also called vector quantization), while "mapping" automatically classifies a new input vector.A self-organizing map consists of components called nodes or neurons. Associated with each node are weight vectors of the same dimension as the input data vectors, and a position in the map space. The usual arrangement of nodes is a two-dimensional regular spacing in a hexagonal or rectangular grid. The self-organizing map describes a mapping from a higher-dimensional input space to a lower-dimensional map space. The procedure for placing a vector from data space onto the map is to find the node with the closest (smallest distance metric) weight vector to the data space vector.While it is typical to consider this type of network structure as related to feedforward networks where the nodes are visualized as being attached, this type of architecture is fundamentally different in arrangement and motivation.Useful extensions include using toroidal grids where opposite edges are connected and using large numbers of nodes.It has been shown that while self-organizing maps with a small number of nodes behave in a way that is similar to K-means, larger self-organizing maps rearrange data in a way that is fundamentally topological in character.It is also common to use the U-Matrix. The U-Matrix value of a particular node is the average distance between the node's weight vector and that of its closest neighbours. In a square grid, for instance, we might consider the closest 4 or 8 nodes (the Von Neumann and Moore neighbourhoods, respectively), or six nodes in a hexagonal grid.Large SOMs display emergent properties. In maps consisting of thousands of nodes, it is possible to perform cluster operations on the map itself.

Aim:Write a Matlabprogramfordrawingfeature maps(KohonenSelfOrganizingFeature maps)in 1-Dimensionalview.

Program

```
% Demonstration of Self Organizing Feature Maps using  Kohonen's Algorithm
clear;
clc;
czy = input('initialisation? Y/N [Y]: ','s');
    if isempty(czy), czy = 'y' ;  end
if (czy == 'y') | (czy == 'Y'),
clear
% Generation of the input training patterns.First, the form of the input domain is selected:
indom = menu('Select the form of the input domain:',...
        'a rectangle', ...
        'a triangle', ...'
        'a circle', ...
        'a ring'  , ...
        'a cross' ,...
        'a letter A');
    if isempty(indom), indom = 2; end
% Next, the dimensionality of the output space, l, is selected.
% The output units ("neurons") can be arranged in a linear, i.e. 1-Dimensional way, or in a rectangle,
i.e., in a 2-D space.
el = menu('Select the dimensionality of the output domain:',...
            '1-dimensional output domain', ...
            '2-dimensional output domain');
    if isempty(el), el = 1; end
 m1 = 12 ; m2 = 18;  % m1  by  m2  array of output units
 if (el == 1), m1 = m1*m2 ; m2 = 1 ; end
 m = m1*m2 ;
 fprintf('The output lattice is %d by %d\n', m1, m2)
 mOK = input('would you like to change it? Y/N [N]: ','s');
```

```
if (mOK == 'y') | (mOK == 'Y')
  m = 1 ;
  while ~((m1 > 1) & (m > 1) & (m < 4000))
    m1 = input('size of the output lattice: \n  m1 = ') ;
    if (el == 2)
     m2 = input('m2 = ') ;
    end
    m = m1*m2 ;
  end
end
fprintf('The output lattice is %d by %d\n', m1, m2)
% The position matrix V
if el == 1
   V = (1:m1)' ;
else
   [v1 v2] = meshgrid(1:m1, 1:m2); V = [v1(:) v2(:)];
end
% Creating input patterns
N = 20*m ;    % N is the number of input vectors
X = rand(1, N)+j*rand(1, N) ;
ix = 1:N;
 if (indom == 2),
  ix = find((imag(X)<=2*real(X))&(imag(X)<=2-2*real(X))) ;
  elseif (indom == 3),
    ix = find(abs(X-.5*(1+j))<= 0.5) ;
  elseif (indom == 4),
    ix = find((abs(X-.5*(1+j))<= 0.5) & (abs(X-.5*(1+j)) >= 0.3)) ;
  elseif (indom == 5),
    ix = find((imag(X)<(2/3)&imag(X)>(1/3))| ...
              (real(X)<(2/3)&real(X)>(1/3))) ;
  elseif (indom == 6),
    ix = find((2.5*real(X)-imag(X)>0 & 2.5*real(X)-imag(X)<0.5) | ...
            (2.5*real(X)+imag(X)>2 & 2.5*real(X)+imag(X)<2.5) | ...
          (real(X)>0.2 & real(X)<0.8 & imag(X)>0.2 & imag(X)<0.4) );
end
X = X(ix); N = length(X);
figure(1)
clf reset,  hold off,  % resetting workspace
plot(X, '.'), title('Input Distribution')
% Initialisation of weights:
W = X(1:m).' ;  X = X((m+1):N) ; N = N-m ;
% as a check, the count of wins for each output unit is calculated in the matrix  "hits".
hits = zeros(m,1);
% An Initial Feature Map
% Initial values of the training gain, eta, and the spread, sigma of the neighborhood function
eta = 0.4 ;   % training gain
sg2i = ((m1-1)^2+(m2-1)^2)/4 ; % sg2 = 2 sigma^2
sg2 = sg2i ;
figure(2)
clf reset
plot([0 1],[0 1],'.'), grid, hold on,
if el == 1
  plot(W, 'b'),
else
  FM = full(sparse(V(:,1), V(:,2), W)) ;
  plot(FM, 'b'),  plot(FM.', 'r') ;
```

```
 title(['eta = ', num2str(eta,2), ...
       '  sigma^2 = ', num2str(sg2/2,3)])
 hold off,
 %  end of initialisation
else  % continuation
 eta = input('input the value of eta [0.4]: ') ;
   if isempty(eta), eta = 0.4; end
 sg2 = input(['input the value of 2sigma^2 [', ...
        num2str(sg2i), ']: ']) ;
   if isempty(sg2), sg2 = sg2i; end
end
reta = (0.2)^(2/N); rsigma = (1/sg2)^(2/N) ;
%  main loop
frm = 1;
for n = 1:N
 % for each input pattern, X(n),  and for each output unit which store the weight vector W(v1, v2),
the distance between
 % X(n) and  W  is calculate  WX = X(n) - W ;  Coordinates of the winning neuron, V(kn, :), i.e.,the
neuron for which
 % abs(WX) attains minimum
 [mnm kn] = min(abs(WX));  vkn = V(kn, :) ;
 hits(kn) = hits(kn)+1; % utilization of neurons
 % The neighborhood function, NB, of the "bell" shape, is centered around the winning unit V(kn, :)
 rho2 = sum(((vkn(ones(m, 1), :) - V).^2), 2) ;
 NB = exp(-rho2/sg2) ;
 % Finally, the weights are updated according to the Kohonen learning law:
 W = W + eta*NB.*WX ;
 % Values of  "eta", and "sigma" are reduced
 if (n<N/2), %ordering and convergence phase
   sg2 = sg2*rsigma;
 else
   eta = eta*reta;
 end
 % Every  100  updates, the feature map is plotted
 if rem(n, 10) == 0
   plot([0 1],[0 1],'.'), grid, hold on,
   if el == 1
     plot(W, 'b'), plot(W, '.r'),
   else
     FM = full(sparse(V(:,1), V(:,2), W)) ;
     plot(FM, 'b'),  plot(FM.', 'r') ;
   end
   title(['eta = ',num2str(eta,2), ...
         '  sigma^2 = ', num2str(sg2/2,3), ...
         '   n = ', num2str(n)])
   hold off,
end
 if sum(n==round([1, N/4, N/2, 3*N/4 N]))==1
   print('-depsc2', '-f2', ['Jsom2Dt', num2str(frm)])
   frm = frm+1;
 end
end
% Final presentation of the result
plot([0 1],[0 1],'.'), grid, hold on,
if el == 1
 plot(W, 'b'), plot(W, '.r'),
```

```
  FM = full(sparse(V(:,1), V(:,2), W)) ;
  plot(FM, 'b'),  plot(FM.', 'r') ;
end
title('A Feature Map'), hold off
```

Exercise 10: UseKohonenSelfOrganizingfeaturemaptoClusterthe vectors(assumefourbinaryvectors)usingowninitial weights(to be assumed)and learningrate(tobe assumed).

Description:

Learning Vector Quantization can be understood as a special case of an artificial neural network, more precisely, it applies a winner-take-all Hebbian learning-based approach. It is a precursor to self-organizing maps (SOM) and related to neural gas, and to the k-Nearest Neighbor algorithm (k-NN). LVQ was invented by Teuvo Kohonen.An LVQ system is represented by prototypes W=(ω(i),…..ω(n)) which are defined in the feature space of observed data. In winner-take-all training algorithms one determines, for each data point, the prototype which is closest to the input according to a given distance measure. The position of this so-called winner prototype is then adapted, i.e. the winner is moved closer if it correctly classifies the data point or moved away if it classifies the data point incorrectly.

Aim: Write a Matlab program to use Kohonen Self Organizing feature map to Clusterthe vectors (assumefourbinaryvectors)usingowninitial weights(to be assumed)and learningrate(tobe assumed).

Program

```
%Learning Vector Quantization
clc;
clear;
s=[1 1 0 0;0 0 0 1;0 0 1 1;1 0 0 0;0 1 1 0];
st=[1 2 2 1 2];
alpha=0.6;
%initial weight matrix first two vectors of input patterns
w=[s(1,:);s(2,:)]';
disp('Initial weight matrix');
disp(w);
%set remaining as input vector
x=[s(3,:);s(4,:);s(5,:)];
t=[st(3);st(4);st(5)];
con=1;
epoch=0;
while con
   for i=1:3
      for j=1:2
         D(j)=0;
         for k=1:4
            D(j)=D(j)+(w(k,j)-x(i,k))^2;
         end
      end
      for j=1:2
         if D(j)==min(D)
            J=j;
         end
      end
      if J==t(i)
         w(:,J)=w(:,J)+alpha*(x(i,:)'-w(:,J));
      else
```

```
      end
   end
   alpha=0.5*alpha;
   epoch=epoch+1;
   if epoch==100
      con=0;
   end
end
disp('Weight Matrix after 100 epochs');
disp(w);
```

Output

Initial weight matrix

```
     1     0
     1     0
     0     0
     0     1
```

Weight Matrix after **100** epochs

```
    1.0000      0
    0.2040   0.5615
         0   0.9584
         0   0.4385
```

UNIVERSITY OF ENGINEERING AND MANAGEMENT, JAIPUR

Course Description

Title of Course: Seminar
Course Code: MCSE181
L-T-P scheme: 0-2-0 **Course Credit: 1**

The overall aim of the seminar series is to help develop an emerging field at the intersection of multi-disciplinary understandings of culture and education. It will build on the existing body of work on education and culture, but its aim is explore and develop new perspectives in this area.
The objectives of the six exploratory seminars are:

- **to explore new research from a range of academic disciplines which sheds light on the questions outlined above**
- **to showcase cutting edge research on education and culture from outstanding academic researchers from the UK and internationally**
- **to bring together seminar participants from different disciplines such as Sociology, Philosophy, Psychology, Human Geography, Media Studies as well as Education and Cultural Studies**
- **to encourage and financially support the participation of PhD students**
- **to actively involve practitioners and users from each venue**
- **to engage a core group of policy makers**
- **to use the seminars to develop links between academics and stakeholders in the arts, library, media, community and educational sectors**