Course Code CSE3003	Operating Systems	Course Type LTP	Credits 4

Course Objectives:

- **To study and apply concepts relating to operating systems, such as concurrency and control of asynchronous processes, deadlocks, memory management, processor scheduling, File System, Security and Virtualization.**

Course Outcomes:

Demonstrate an understanding of:

- Basics of operating system, its structures and services
- The differences between processes and threads.
- The different process or thread synchronization methods and the tradeoffs between them
- The different memory management and Scheduling techniques used in Operating Systems
- Deadlock and solving its related issues
- Various I/O management techniques used in Operating Systems.
- File system and its implementation in storage device
- The tradeoffs in design and implementation concepts used in the development of Operating Systems
- Efficient use of hardware through Virtualization

Student Outcomes (SO): b,h,i,l

Unit No	Unit Content	No. of hours	SOs
1	**Basic of Operating System and Its Structures** **I**ntroduction: Computer System Organization-Arhitecture-Structure-Operations. Management:Process-Memory-Storage. Structures:Services-System Interface- System Calls- System Program-Design-structure	9	b
2	**Process and Threads** Introduction to Process – Scheduling – Operations-Interprocess Communication. Synchronization: Critical Section-Hardware-Mutex- Semaphore –Monitors. Threads: Multithreading Models-Thread Library- Issues	9	i,l
3	**Processor Scheduling and Deadlocks** CPU Scheduling :Scheduling Criteria- Algorithms-Evaluation. Deadlocks: Principles- Prevention- Avoidance-Detection-Recovery	9	i,l
4	**Memory and Storage Management** Main Memory:Swapping-Contigious Memory Allocation – Segmentation – Paging. Virtual Memory:Demand Paging- Page Replacement Algorithm. Secondary Storage: Disk Scheduling-Disk Management- RAID	9	b,h
5	**File System,I/O and Security** File Systems: Concepts- Structure-Allocation Methods. I/O	7	h,l

	Systems:Harware-Interface-Transformation.Security and Protection:Access Matrix- Access Control-Program Threats-Cryptography-Defense Mechanism.Guest Lecture on		
6	Contemporary Topics (Virtualization and Cloud Environment)	**2**	
	Total Hours:	**45**	

Mode of Teaching and Learning: Flipped Class Room, Activity Based Teaching/Learning, Digital/Computer based models, wherever possible to augment lecture for practice/tutorial and minimum 2 hours lectures by industry experts on contemporary topics

Mode of Evaluation and assessment:

The assessment and evaluation components may consist of unannounced open book examinations, quizzes, student's portfolio generation and assessment, and any other innovative assessment practices followed by faculty, in addition to the Continuous Assessment Tests and Final Examinations.

Text Books:	
1.	Abraham Silberschatz, Peter Baer Galvin and Gre Gagne, Operating System Concepts, Wiley Publication, Ninth Edition 2012
2.	Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau, Operating Systems, Three Easy Pieces, Arpaci-Dusseau Books, Inc (2015).

Reference Books:	
1.	William Stallings, Operating Systems Internal and Design Principles, Pearson, Seventh Edition, 2017
2.	Sibsankar haldar,Alex A Aravind, Operating Systems, Pearson, Second Edition, 2016

Indicative List of Experiments

No.	Description of Experiment	SO
1	Study of Hardware/Software requirement of various operating system. (**I. (a)** Study of hardware and software requirements of different operating systems (UNIX, LINUX, WINDOWS XP, WINDOWS 7/8). (1 Lab session) **(b)** Execute various UNIX system calls for (1 Lab session) 1. Process management 2. File management 3. Input / Output System calls **IMPLEMENTATION DETAILS AND ASSUMPTIONS:** (i) The OBJECTIVE of this practical is to obtain general overview of various popular OS . (a) Their development and distribution (b) Compatibility (c) Security issues and Thread detection and its solution (d) The GUI etc.. (ii) Along with the above mentioned activities, execution of various UNIX commands is also helpful: cat, cd, cp, chmod, df, less, ls, mkdir, more, mv, pwd, rmdir, rm, man, uname, who, ps, vi, cal, date, echo, bc, grep	i,l
	(CPU Scheduling Policies)	i,l

2	**I.** Implement CPU scheduling policies : (2-3 Lab sessions) **(a)** SJF **(b)** Priority **(c)** FCFS **(d)** Multi-level queue **IMPLEMENTATION DETAILS AND ASSUMPTIONS:** **INPUT/s:** (i) The number of processes/jobs in the system (computed through random functions in C) (ii) The CPU Burst (based on past history), priority (initially, compute through random function), arrival time of process. **STEPS TO PERFORM:** **(a)** For **SJF** algorithm, (i) We randomly generate the number of jobs. There must be a limit on the number of jobs in a system. (ii) The execution time of the generated jobs is also not known. Here, we are generating the CPU burst of each job making use of the past history. (iii) All the jobs are then arranged in a queue where searching is done to find the one with the least CPU burst. There may be two jobs in queue with the same execution time then FCFS approach is to be performed. **Case a)** If the algorithm is **non preemptive** in nature, then the newly arriving job is to be added to the job queue even though it is of lesser execution time than the one running on the processor. **Case b)** Otherwise **preemption** is performed. **(b)** For **Priority scheduling**, (i) We again prefer to compute the CPU burst from past history. (ii) Priority may be assigned on the basis of their CPU burst (simplest approach) (iii)Priority may be assigned through some random function (within a specific range say 1 to 100). (iv) Priority has to be changed dynamically (to perform aging in order to avoid starvation). **Priority (preemption)** and **priority (non preemption)** nature of priority scheduling is performed. **(c)** The **FCFS scheduling** is performed on the basis of arrival time irrespective of their other parameters. **(d)** In **multi-level queue scheduling**, different queues are to be created. **OUTPUT/s:** The average throughput, waiting time of process/s	
3.	**(File Storage Allocation Techniques)** **I.** Implement file storage allocation techniques: **(a)** Contiguous (using array) (1 Lab session) **(b)** Linked –list (using linked list) (1 Lab session) **(c)** Indirect allocation (indexing) (2 Lab sessions at max.) **IMPLEMENTATION DETAILS:** **INPUT/s :** (i) Free storage blocks to allocate storage to files.	i,l

	STEPS TO PERFORM: Before performing storage allocation, we need to perform the following common steps: (i) During program execution, we need an illusion that we have a large disk space for allocation strategies. (ii) We take a big array of structure (representing storage space in terms of sectors). This space is equal to the disk size available for storage. (iii) Next step is to see free space list. The free space list is not entirely free. Some portion of this list is occupied by some OS or user files/data. (iv) The free-space is to be maintained in a suitable manner as per allocation strategy. (v) Upon the above formed disk space and free space list, we then implement the above mentioned techniques. (vi) Symbolic file name and size will be inputted by the user and accordingly the assumed disk space as well as free space list will be updated. (a) Contiguous allocation strategy is implemented using array data structure. (b) Linked list allocation technique is implemented using linked list. (c) Indirect allocation is performed using indexing concept. **OUTPUT/s:** Files/ Programs are allocated storage space through appropriate storage allocation techniques.	
4.	**Contiguous Allocation Techniques** **I.** Implementation of Contiguous allocation techniques: **(a)** Worst-Fit **(b)** Best-Fit **(c)** First-Fit **IMPLEMENTATION DETAILS:** **INPUT/s:** (i) Free space list of blocks from system (as created in experiment 3) (ii) List processes and files from the system (as in experiment 3) **STEPS TO PERFORM**: (i) We consider the same free space list and files/processes as created in experiment 3 for our system. (ii) Implement the above mentioned three contiguous allocation techniques. Also, the free space list is updated from the free blocks left out after performing allocation. **(a) Worst-fit**: In worst fit technique largest available block/partition which will hold the page is selected. Blocks are sorted according to their size in descending order. **(b) Best-fit**: Best-fit is one of the optimal technique in which page is stored in the block/partition which is large enough to hold it. Blocks are sorted according to their size in ascending order. **(c) First-fit**: In first-fit technique page is stored in the block which is encountered	i,l

	first that is big enough to hold it. (iii) Also, the free space list is updated from the free blocks left out after performing allocation. **OUTPUT/s:** Processes and files allocated to free blocks. List of processes and files which are not allocated memory. The remaining free space list left out after performing allocation.	
5	**External and Internal Fragmentation** Calculation of external and internal fragmentation. **IMPLEMENTATION DETAILS:** **INPUT/s:** (i) Free space list of blocks from system (as in experiment 3). (ii) List processes and files from the system (as in experiment 3). **STEPS TO PERFORM:** (i) Completing experiment 4, we end up getting list of allotted files, remaining part of allotted block and blocks which cannot be allotted (ii) After implementing each allocation algorithm, list the amount of free space blocks left out after performing allocation. (iii) When a block which is not at all allocated to any process or file, it adds to external fragmentation. (iv) When a file or process is allocated the free block and still some part of it is left unused, we count such unused portion into internal fragmentation. **OUTPUT/s:** Processes and files allocated to free blocks. From the list of unused blocks, we determine the count of total internal and external fragmentation.	i,l
6	**External and Internal Fragmentation** Implementation of Compaction for the continually changing memory layout and calculate total movement of data. **IMPEMENTATION DETAILS:** Compaction is a technique used to remove internal fragmentation. Assumption that has to be taken into consideration is that the files must be inserted and deleted continuously. User must provide memory image at different instances of time. In the experiment no 5, we have obtained total internal and external fragmentation. To the above practical, we continue performing the compaction. Hereby, this activity is left for students to think and perform compaction to the above practical.	i,l
7.	**Resource Allocation Graph (RAG)** Implementation of resource allocation graph (RAG). **IMPLEMENTATION DETAILS:** **INPUT/s:** (i) List of resources (ii) Instance of each resource (for case 2 only) (iii) List of processes (iv) Resource allocated by each process **STEPS TO PERFORM:** (i) List the processes and resources.	i,l

	(ii) We read input from user for each [Pi , Ri] and also how many instances of each resource to a particular process (for multiple instances case). (iii) While user completes the input, we end up constructing adjacency matrices/ list. Two cases to be considered: **Case 1–** Each resource has single instance (simpler problem). **Case 2–** Multiple instances of resources (complex problem). **a.** Methods used for representing graph: **(i) Adjacency matrix**: A 2-D array of size N x N where N is the number of vertices in the graph (includes processes and resources). For each adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Since resource allocation graph is directed graph, hence it is not necessary to be symmetric. **(ii) Adjacency list:** An array of linked list is used. Size of the array is equal to number of vertices (processes) in the graph. An entry arr[i] represents the linked list of vertices (resources requested by process) adjacent to the ith vertex. **OUTPUT/s:** Output is a Resource allocation graph through matrices/list.	
8.	**Bankers Algorithm** Implementation of Banker's Algorithm. **IMPLEMENTATION DETAILS:** **INPUT/s:** Basic input required to implement the Banker's Algorithm: (i) Available (ii) Max (iii) Allocation **STEPS TO PERFORM:** (i) Perform Banker's algorithm when a request for R is made. (ii) Compute Need[i,j] = Max[i,j] - Allocation[i,j]. (iii) Update accordingly. Once the resources are *allocated*, check to see if the system state is safe. If unsafe, the process must wait and the old resource-allocated state is restored. **OUTPUT/s:** Detection process specifies if a deadlock is present in system with listed processes and their needs or not.	i,l
9.	**Wait Graph** Conversion of resource allocation graph (RAG) to wait-for-graph (WFG) for each type of method used for storing graph. **IMPLEMENTATION DETAILS:** One such deadlock detection algorithm makes use of a wait-for graph to track which other processes a process is currently blocking on. In a wait-for graph, processes are represented as nodes, and an edge from process Pi to Pj implies Pj is holding a resource that Pi needs and thus Pi is waiting for Pj to release its lock on that resource. There may be processes waiting for more than a single resource to become available. Graph cycles imply the possibility of a deadlock. **INPUT/s:** Output of experiment 7 as Resource allocation graph (RAG) through adjacency	i,l

	matrices/list. **STEPS TO PERFORM:** (i) Identify the waiting processes in the RAG. (ii) Accordingly draw Wait-for graph for the given RAG. (iii) To draw it as graphical representation, we introduce graphics.h and work in graphics mode. (iv) Geometric images are entered by entering graphics, providing with parameter and closing graphics. (v) We now identify circular chain of dependency (i.e., appearance of loops in the graph) **OUTPUT/s:** (i) The wait-for-graph(graphical representation). (ii) Also, check presence of loop to detect if loop is present	
10.	**Inter process Communication – Semaphore** Implement the solution for Bounded Buffer (Producer-Consumer) problem using inter process communication technique – Semaphores. **II.** Implement the solution for Readers-Writers problem using inter process communication technique – Semaphores. **III.** Implement the solution for Dining-Philosopher problem using inter process communication technique – Semaphores. **IMPLEMENTATION DETAILS:** (i) For programming this problem, we use JAVA (multi-threading concept) for implementing the synchronization problem using semaphores. (ii) Our main focus is to obtain three conditions of (a) mutual exclusion (b) progress (c) bounded wait. (iii) Implement semaphore concept considering above mentioned problem. **OUTPUT/s:** Synchronization of the problem satisfying conditions of mutual exclusion, progress and bounded wait.	i,l
11.	**FORK and JOIN construct** **I.** Write a program where parent process take average of the odd numbers and child process will take the average of even numbers present in a given Aadhar number of a person. Use FORK and JOIN construct. **II.** Write a program where parent process finds additive primes and child process finds circular prime for a given prime list array. Use FORK and JOIN construct. **IMPLEMENTATION DETAILS:** (i) Parent process and child process can communicate with each other with the help of shared memory. Parent process and child process both will work on the data available in that shared memory and according to them provide their outputs. Two cases will arise according to their sequence of termination: **Case a)** Child process waits until parent process terminates. **Case b)** Parent process waits until child process terminates. (ii) If child process terminates before parent process, process execution will be	i,l

	unsuccessful. (iii) At a time only one process is executing. Control will not be transferred to another process until one process does not complete its execution i.e. in non-preemptive manner. According to their termination proper message must be displayed.	

Recommendation by the Board of Studies on	25/06/2018
Approval by Academic council on	18/07/2018
Compiled by	*Dr S Raju and Dr R Ganesan*