OPERATING SYSTEMS

1. Introduction

Objective: In this course, a detailed view of the kernel is given. This includes a detailed description of the way system calls work, the extended process state diagram in process management, detailed file system management description including the
data structures such as *inodes* used to maintain metadata and introduction to device drivers as part of the I/O subsystem.

This course excludes user space application development programming such as using IPC or thread-based programming. This is included in the *Network Programming* course.

Credits: 3-0-1

2. Prerequisites

C programming, Data and File Structures.

3. Course Outline

UNIT - I: Introduction and Operating System Structures
Operating Systems Functionality, Computer Organization and Architecture, OS Operations, Kernel Data Structures, OS Services, User interfaces to OS, Programmer interfaces to OS, OS Structure, System Boot.

UNIT - II: Process and Thread Management
Process Concept, Process operations, Process Scheduling, Extended Process State Diagram (To be done from Stallings, Operating Systems: Internals and Design Principles), Process Context Switch in detail including System Call inter- face and implementation (To be done from Crowley: Operating Systems: A design-oriented approach), Interprocess Communication: Pipes, Named Pipes, Shared Memory, Process Synchronization: Signals, Mutexes, Semaphores, Monitors (To be done from Silberschatz et al. and Stevens), Thread Management: thread creation, thread scheduling, thread synchronization, Deadlocks: Resource Allocation Graphs, deadlock detection, prevention and avoidance, recovery from deadlock.

UNIT - III: Memory Management
Memory allocation techniques: paging and segmentation, Swapping, structure of the page table, Virtual memory: demand paging, copy-on-write, Page replacement, allocation of frames, kernel memory allocation, thrashing, memory-mapped files, Translation-Lookaside Buffer (TLB), multiprocessor concerns.

UNIT - IV: File System Management
Disk management: formatting, boot block, swap-space management, RAID structure, Disk scheduling algorithms: elevator, C-SCAN, File concept, Access methods,Directory structure, File system mount and unmount operations, file sharing, protection, file system structure, file system implementation: file system metadata storage structures such as inode (To be done from Bach: The Design of the Unix OS), allocation methods, free space management, efficiency and performance including disk cache and recovery from failures.

UNIT - V: I/O Management
I/O devices: polling, interrupt-driven, DMA, Application I/O interface: character and block devices, network devices, clocks and timers, nonblocking and asynchronous I/O, vectored I/O, Kernel I/O interface: I/O scheduling, Buffering, Caching, Spooling and device reservation, error handling, I/O protection, Kernel data structures Transforming I/O requests to hardware operations, Performance.

UNIT - VI: Case Studies1
Linux, Unix: Solaris/AIX, Windows 7.

4. Reading Material

Text Books
Abraham Silberschatz, Peter Baer Galvin and Greg Gagne. Operating System Concepts , 9th edition, Wiley

Reference Books

1. Charles Crowley. Operating Systems: A Design-Oriented Approach, Prentice-Hall India.
2. W. Richard Stevens, . Advanced Programming in Unix Environment, Pearson Education.
3. W. Richard Stevens. Unix Network Programming, vol. 2, Pearson Education.
4. William Stallings. Operating Systems: Internals and Design Principles, Pearson Education.
5. Maurice J. Bach. The Design of the Unix Operating System, Prentice-Hall India.
6. Robert Love. Linux Kernel Development, Pearson Education.
7. Thomas Anderson and Michael Dahlin. Operating Systems: Principles and Practice, 2nd edition, Recursive Books.

Suggested Assignments

1. Modify the kernel to include the statement "Hello, World", compile the kernel and modify the boot loader to add the new version.
2. Compare the popular file systems: ext3, NTFS and XFS.
3. Implement small modifications of Producer-Consumer problem.
4. Implement a user level ps command by walking through the /proc directory.
5. Implement page tables and virtual-physical address mapping using the page tables. In this, the input can consist of a file containing the processor arch. (16/32/64-bit), page size, available RAM. For each process the file will contain the size of the executable and/or page no. and permissions on it and the frame start address. Then a set of virtual addresses with the operation (r/w/x) are input to the program for which the corresponding physical address must be returned if it is a valid virtual address and also whether the operation attempted is legal as per the permissions on the page.
6. Go through the /etc/fstab and manipulate it to mount partitions with different settings and/or use mount/umount commands to mount and unmount partitions and understand the concept of logical volumes.
7. Implement a simple file system such as FAT using FUSE API.
8. Implement a simple software device driver.