CL020 - Advanced Linux and UNIX Programming

Course Description:

In-depth training for software developers on Linux and UNIX system programming facilities. Learn how to develop sophisticated multiprocess applications using system calls and library routines.

Audience:

Application developers who will be writing advanced programs on Linux and UNIX.

Prerequisites:

Fundamentals of UNIX or Fundamentals of Linux, C Programming, and Advanced C Programming. Strong C programming skills are required for this course.

Course Contents

UNIX Standards

- Brief History of UNIX
- AT&T and Berkeley UNIX Systems
- Major Vendors
- What is a Standard?
- What is POSIX?
- Other Industry Specs and Standards

Files and Directories

- The POSIX.1 Basic File Types
- File Descriptions
- Keeping Track of Open Files
- File Table Entries
- The v-node Structure
- The fcntl Function
- File Attributes
- The access Function
- Link, unlink, remove, and rename Functions
- Functions to Manipulate Directories

System I/O

- Standard I/O vs System I/O
- System I/O Calls
- File and Record Locking

Processes

- What is a Process?
- Process Creation and Termination
- Process Memory Layout
- Dynamic Memory Allocation
- Accessing Environment Variables
- Real and Effective User IDs

Process Management

- Programs versus Processes
- The fork() System Function
- Parent and Child
- The exec System Function
- Current Image and New Image
- The wait() and waitpid() Function
- Interpreter Files and exec

Pipes - Basic IPC

- Interprocess Communication
- FIFOs
- More on FIFO's

Signals

- What is a Signal?
- Types of Signals
- Signal Action
- Blocking Signals from Delivery
- The sigaction() Function
- Signal Sets and Operations
- Sending a Signal to Another Process
- Blocking Signals with sigprocmask()
- Scheduling and Waiting for Signals
- Restarting System Calls (SVR4)
- Signals and Reentrancy

Overview of Client/Server Programming

- Designing Distributed Application
- Clients and Servers
- Ports and Services
- Server Types
- Stateless vs. Stateful Servers
- Concurrency Issues

The Berkeley Sockets API

- Berkeley Sockets
- Data Structures of the Sockets API
- Socket System Calls
- Generic Client/Server Models
- Sample Socket-based Client

Algorithms and Issues in Client Design

- Algorithms Instead of Details
- Client Architecture
- Sockets Utility Functions

TCP Client Algorithm

- TCP Client Implementation
- UDP Client Algorithm
- UDP Client Implementation

Server Design

- Iterative Servers
- Concurrent Servers
- Performance Consideration
- An Iterative Server Design
- A Concurrent Server Design

System V Interprocess Communication

- System V IPC
- The Three System V IPC Facilities
- Common Operation - Get (IPCget)
- Common Operation - Control (IPCctl)
- Calls to Operate on the Facilities
- Commonalities between msg, sem, and shm
- IPC via Message Queues
- IPC via Shared Memory Segments
- Coordinating the Use of Shared Memory
- Semaphore Sets-semget() and semctl() Calls
- Semaphore Sets - the semop() calls
- Shared Memory Coordination Using Semaphores
- IPC Facility Handling ipcs and ipcrm

Date and Time Functions

- Time Representations
- Decoding Calendar Time
- Shorthand Functions - asctime(), ctime()
- Formatting Calendar Time Shared
- Process Times
- The Difference Between clock() and times()
- Berkeley High resolution Timers

Standard I/O

- I/O Calls to manipulate streams
- I/O Calls which perform character I/O
- I/O Calls which perform string I/O
- I/O Calls which perform formatted I/O
- I/O Calls which perform binary I/O