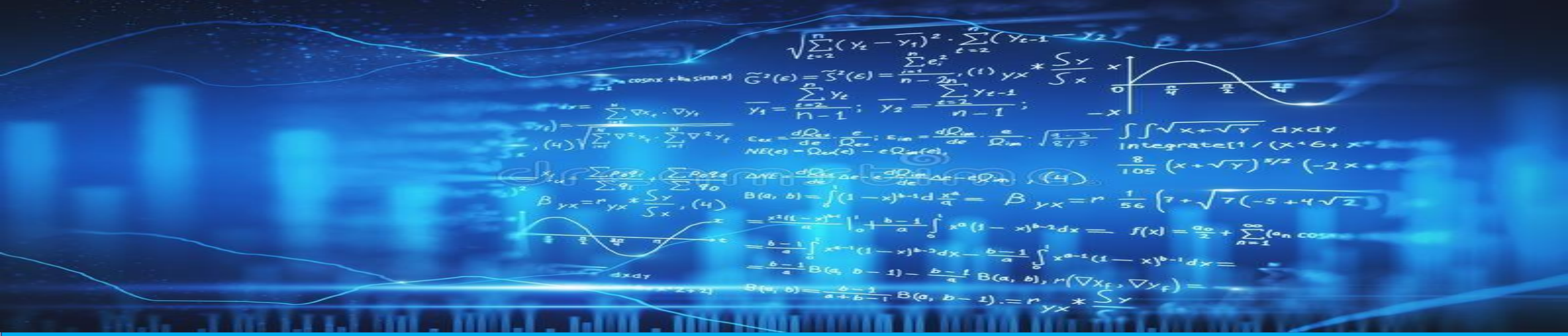


Upscale Your Problem Solving and Embedded Linux Programming Skills with Master Programs

Unparalleled Learning Experience

- ➔ **Advanced Algorithmic Thinking**
- ➔ **System Programming with Linux**
- ➔ **Kernel Programming and Driver Development**
- ➔ **Embedded Linux**

Advanced Algorithmic Thinking

"Advanced Algorithmic thinking gives you deep insights into data structures and algorithmic design patterns which helps you to master key skills required for coding interviews and competitive programming. This course is designed to focus more on problems related to 1D and 2D -Dynamic programming problems, Dynamic tree problems, pattern matching, string problems, and bitwise problems. This approach gives an inquiry starting from a given condition to investigate or demonstrate a fact"

40+ hrs of live interactive sessions which include a conceptual framework and hands-on research examples

Technical Requirements

Desktop/Laptop with the below configuration

RAM: 2GB or more if running on a physical host/ 4GB or more if running as a guest (Using VM Virtual Box)

SSD/HHD: 30 GB of free space

Operating System: Ubuntu 18.04 LTS or above/Windows 8 and above with Internet Access

Hardware: Laptop/Desktop with Ubuntu 18.04 or above/ Windows 8 and above

Course Curriculum

Data Structures

- 1D and 2D Array
- List
- Stack and Queue
- Set and Multiset
- Priority queues
- Set and Multiset
- Map and Multimap
- Sorted set and sorted Multiset
- Sorted Map and Sorted Multimap
- Disjoint set
- Bitset
- Trie
- Suffix array and suffix trie
- Red-Black Trees

Algorithmic Design Patterns

- Adhoc Thinking
- Recursive Thinking
- Divide and Prune thinking
- Divide and conquer thinking
- Backtrack thinking
- Greedy thinking

Anatomy of Algorithms

- Time and space complexity analysis
- Best case, Worst case, Average case, and Amortized analysis
- Big (O), Theta (Θ), Small (0), and Omega (Ω) Notations

Tackle Problems on

- Search Problems
- String Problems
- Graph Problems
- Dynamic programming problems (1-D,2-D and Tree)
- Greedy Problems
- Tree Problems
- Backtracking Problems
- Bitwise problems
- Pattern Matching Problems

System Programming with Linux

"System programming with Linux gives Participants deep insights into the operating system concepts, Linux system architecture, and low-level interfaces required to build system-level applications on Linux. This module is carefully designed with practical exercises that provide participants with the knowledge required to write complex systems, networks, and multithreaded applications"

45+ hrs of live interactive sessions which include a conceptual framework and hands-on research examples

Technical Requirements

Desktop/Laptop with the below configuration

RAM: 2GB or more if running on a physical host/ 4GB or more if running as a guest (Using VM Virtual Box)

SSD/HHD: 30 GB of free space

Operating System: Ubuntu 18.04 LTS or above with Internet Access

Hardware: Laptop/Desktop with Ubuntu 18.04 or above

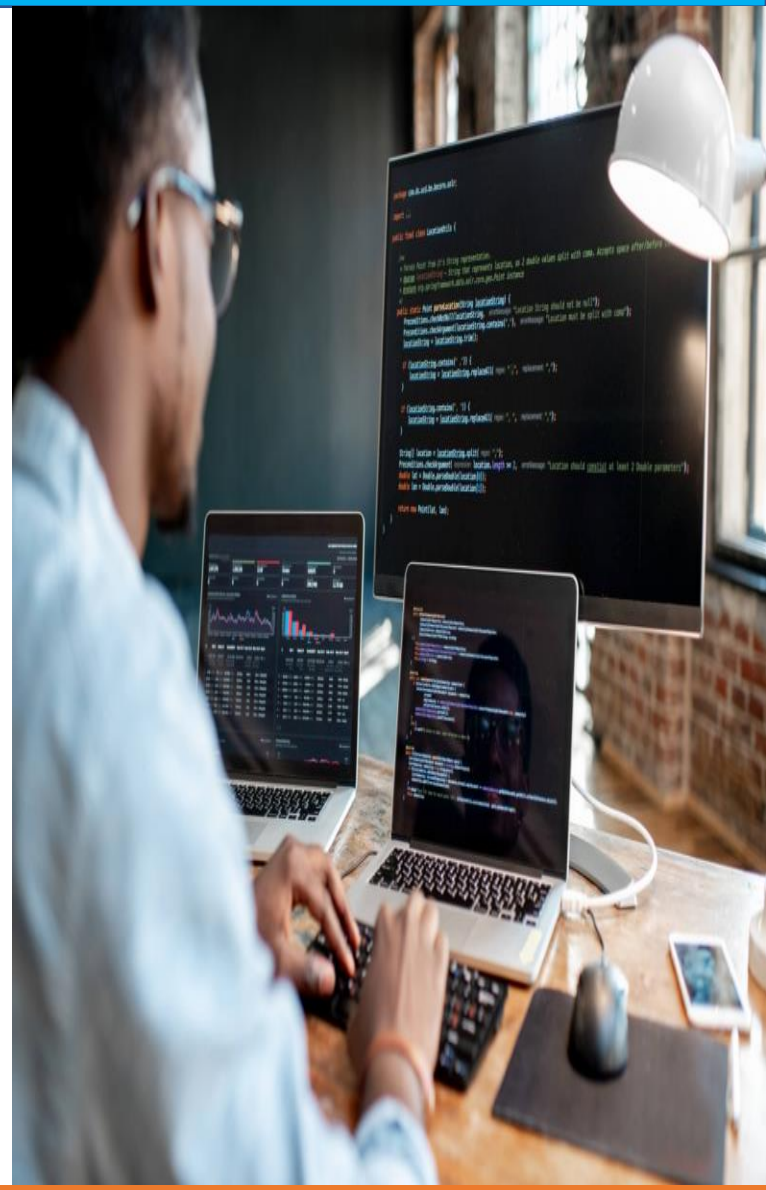
Course Curriculum

Fundamental Concepts

- Library Fundamentals
- Static Linker and Dynamic Linker
- Creation and use of Static and dynamic libraries
- Constructors and destructors
- ELF and Program Execution
- *ldconfig*
- Dynamically Loaded Libraries
- GOT, PLT, and Lazy Binding
- ABI, SYSV

Memory Management

- Virtual addressing (Virtual Memory)
- No VM VS Virtual Memory
- Benefits of using VM
- Process Memory Layout
- Segments or Mappings
- Stack allocation and deallocation
- Dynamic memory allocation and deallocation
- Demand paging, Locking Memory, and Memory Protection
- Linux Common Memory Issues and Test cases
- GDB, Valgrind, and Sanitizer tools



Course Curriculum

File I/O

- Over View
- *Open(), create(), read(), write(), and close ()*
- File Descriptor and Open file relation
- Accessing file attributes
- Modifying File Offset
- File mode
- Changing file attributes

Advanced File I/O

- Scatter – Gather I/O
- MT app file I/O
- File I/O Via Memory Mapping
- DIO and AIO
- Multiplexing or Async blocking I/O or Alternative I/O Models
- Linux I/O architecture
- Race Hazards and File Locking

Process Management

- Converting a Program into a Process
- Process IDs
- Memory Layout of a Process and Process Address Space
- Kernel Process Address Space
- Process Scheduler
- Process Creation and Termination Calls
- Executing Programs

Signals

- Overview of Signals
- Signal Disposition
- Useful Signal-Related Functions
- Signal Types and Categories
- Generation of Signal and Delivery
- Signal Handler and Designing Async Signal Handler
- Process Communication Using Signals

Course Curriculum

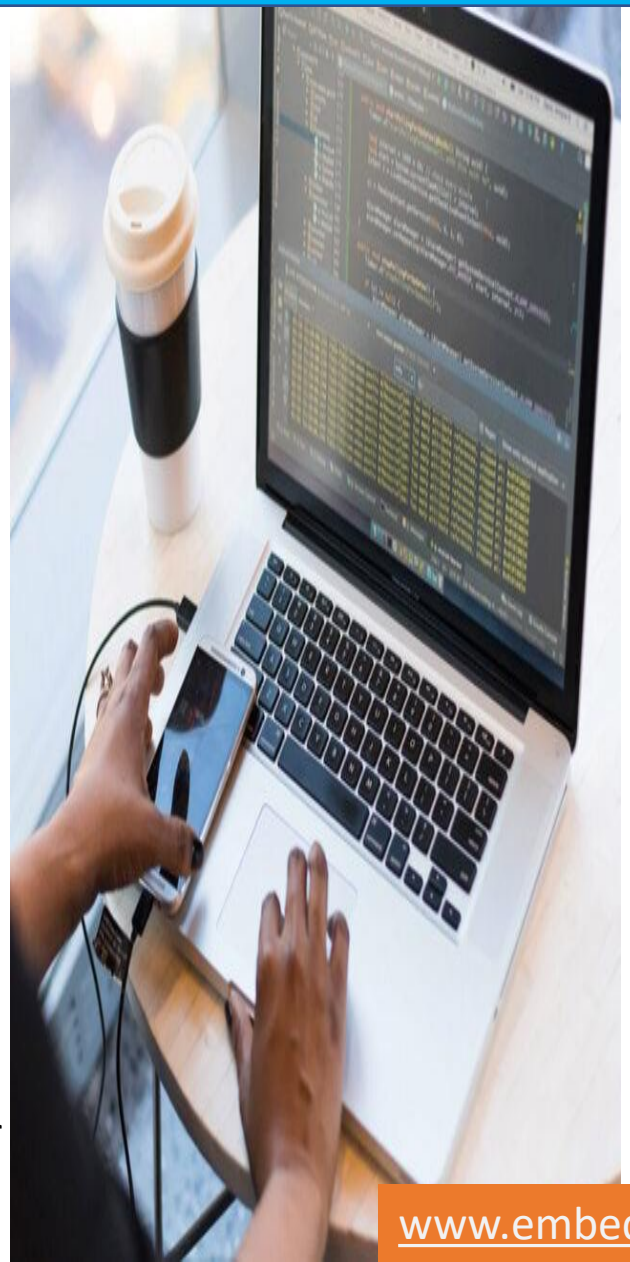
Timers

Threads

- Concurrency and Need of Concurrency
- Pthreads Library
- Thread Creation and Termination
- Thread ID’s
- Joining a Thread
- Detaching a Thread and Thread Attributes
- Signals and Threads
- Threads and Process Control

Synchronization

- Shared Resources and Critical Section
- Atomic Operations
- Mutex and Insights of Mutex
- Locking, Unlocking and Deadlock
- Mutex vs Spinlocks
- Reader-Writer Locks and Producer Consumer
- Condition Variables



IPC

Pipes
FIFO
POSIX Semaphores
POSIX Shared Memory
Unix Domain Sockets

Internet Domain Sockets

Introduction to internet Domain Sockets
Data-representation Issues
Loopback and Wildcard Addresses
Host Addresses and Port Numbers
Host and service conversion
Internet Domain Sockets Example
Additional Sockets System Calls

Kernel Programming and Driver Development

"Kernel Programming and Driver Development help participants in apprehending the essentials of key internal topics such as kernel architecture, memory management, CPU scheduling, and kernel synchronization core kernel subsystems and various design level issues, This module also helps participants implement custom device drivers to support various peripherals on Linux machines"

Technical Requirements

Desktop/Laptop with the below configuration

RAM: 2GB or more if running on a physical host/ 4GB or more if running as a guest (Using VM Virtual Box)

SSD/HHD: 30 GB of free space

Operating System: Ubuntu 18.04 LTS or above with Internet

Hardware: Laptop/Desktop with Ubuntu 18.04 or above

Course Curriculum

Introduction to the Linux Kernel

- Download, Configure Stable Kernel
- Kernel source Layout
- Building Kernel from source
- Create custom Menu entry

Writing Kernel Modules

- K-Space Coding vs U-Space Coding
- Makefile Implementation to Build Modules
- Passing Parameters to Kernel Modules
- Modules Auto Load During Boot

Memory Management

- Kernel Memory allocations
- Kernel Memory Data Structures
- Kernel's OOM Framework
- Kernel Page Tables and Address Translation
- Memory Debugging

CPU Scheduler

- CPU Scheduling on Linux
- Scheduling Policies and Priorities
- Scheduling latencies and Measuring
- CPU Affinity
- Multi Core Scheduling

Kernel Synchronization

- Critical Section
- Atomicity
- Mutex
- Spinlock
- Semaphore
- Deadlock
- Avoiding Deadlock

Course Curriculum

User and Kernel Interaction

- Different Techniques
- ***procfs***
- ***sysfs***
- ***debugfs***
- Netlink Sockets
- ***Ioctl***

Hardware I/O Access

- Memory Mapped I/O
- Generic Technique
- Port Mapped I/O
- Several Use Case

Character Driver

- Design the scull
- Major and Minor Numbers
- Critical Data Structures
- Char Driver Registration
- Driver File Operations (open and release)
- Scull's Memory Usage
- Read and write
- Using the New Device

Interrupt Handling

- Preparing the Parallel Port
- Linux Interrupt Handler
- Implementing a Handler
- Top Bottom Halves
- Sharing the Interrupt
- Interrupt-Driven I/O

Memory Mapping and DMA

- Memory Mapping in Linux
- The mmap Device Operation
- Performing Direct I/O
- Direct Memory Access

Block Drivers

- Registrations
- The Block Driver Operations
- Processing the Requests

Course Curriculum

PCI Drivers

- The PCI Interface
- ISA
- PC/104 and PC/104+
- SBus
- NuBus
- External Buses
- Hands-on with PCI Driver

USB Drivers

- USB Device Basics
- USB and *Sysfs*
- USB Urbs
- Hands-on with USB Driver

Network Drivers

Platform Drivers

Embedded Linux

18+ hrs of live interactive sessions which include a conceptual framework and hands-on research examples

Technical Requirements

Desktop/Laptop with the below configuration

RAM: 2GB or more if running on a physical host/ 4GB or more if running as a guest (Using VM Virtual Box)

SSD/HHD: 30 GB of free space

Operating System: Ubuntu 18.04 LTS or above with Internet

Hardware: BeagleBone Black and EmbedCore Cape board

Course Curriculum

- Introduction to Embedded Linux
- Bootloader
- Build process
- Required packages
- Build root
- Tool chain, kernel and rootfs configuration
- Porting images to target board