

# Embedded Engineering

MIRACLE
TECHNOLOGIES

Ist Floor, B-12, Behind DELITE
Showroom, Adjacent to Metro Station,
Near Gol Chakkar Sector-2, Noida

Ph. : +91-120-3058445
+91-120-3058447
Mobile : 9311305845/6/7

Mail ID :
mcts@miracleindia.com
www.miracleindia.com


## Introduction

This Course has been designed with the vision of creating world class professionals that should provide Linux based solutions in embedded system and real-time technologies.
Our goal is to provide competitive knowledge and embedded solutions, powered by intelligence and innovation. The Embedded Solutions Division offers a wide range of value-added solutions in the Device Design, Device Drivers, Telecoms, Automotive, Consumer, Manufacturing and Convergence market segments.

We have the expertise to develop a complete out of the box embedded system that involves architectural system design and analysis, circuit board design, embedded software design, embedded networking and Communications.
Our solution to the different market segment includes:

- Board design and bring up.
- Boot Loader development / RTOS porting
- Drivers / BSP development for SoC modules and on-board peripherals
- Protocol Stack and Middleware development
- Embedded applications and firmware.

Miracle's highly skilled engineers and consultants complement our in-house development teams, adding just-in-time expertise and resources to help you complete your projects on time and assist in their ongoing maintenance. Our architects have many years of experience in product development, applied research, and technical consulting across the broadest range of embedded platforms and technologies. As a customer, you draw advantage from our highly competent interdisciplinary engagement and delivery teams who are in the forefront of this technically challenging area.

## Abstract

Embedded processors form a core component of an enormous range of systems, from avionics, passenger cars, communication and navigation equipments, defense, and medical equipment, to robotics, gaming devices, DVD players, Electronic gadgets and home appliances. The growth in the power and features of processors has helped to drive the semiconductor industry from start-up some 50 years ago to more than $200 billion in annual revenue today. The embedded market is now estimated to be worth around 100x the "desktop" market, and is forecast to grow exponentially over the next decade.



**Objective**

This short course is designed to ensure that students of Engineering College with academic Capabilities will have the skill set needed to deal with the challenges involved in real-world embedded technologies to meet the needs of industries both today and in the future.

The course considers programming techniques which can help to ensure that single-Processor embedded systems are reliable.The course is taught mainly using the C programming language, with a PC emulated as an embedded device.

**Pre-requisites**

A prior knowledge of a basic Linux commands, general understanding operating system System concepts are assumed.

**Agenda**

The duration of the course will be 122 hours.

- The course is split into five modules.
- After the course is over, you should be able to:
  - Efficient with programming in C.
  - Understand Linux System Concepts in depth.
  - Able to do Linux System Programming
  - Able to write device drivers using Linux.
  - Have architecture level understanding of the ARM processor.
  - Able to configure and customize Linux for ARM based boards.
- The following topics will be taught as part of the course.

## Embedded Engineering

**Modules:**

**Advance C programming**

**Data Structures using C, Analysis and Design of Algorithms**

**Operating System Concepts with Linux System Programming**

**Linux Device Drivers**

**Embedded Linux**


# Advance c programming

**Introduction to C**

- Features of C language
- Broad structure of C
- Data type:
- Escape sequences
- Operators

**Decision Making and Looping**

- Decision Structure
- Looping Structure
- Break and Continue statement

**Functions**

- What are Functions?
- Steps Involved In Processing A Function.
- Recursion Functions.
- Types of functions
- Functions Returning Pointers
- Assignments

**Arrays, Structures and Unions**

- Definition of Arrays
- Comparison between arrays and pointers
- Single Dimension Array
- Two Dimensional Arrays
- Structures & Unions
  - Memory layout
  - Bit fields in structure
  - passing structures in a function
  - Assignments

**Pointers**

- Declaration of pointers
- Rules valid for pointers
- Pointer usage
- Pointers addressing, referencing and dereferencing
- Passing pointers to a function
- Strings with Pointers
- Assignments

**Strings**

- Accessing individual characters
- Printing strings with printf
- Printing strings with puts
- Inputting strings with scanf
- Inputting strings with gets
- String Library (strcpy(),strcat(),strcmp(),strchr(),strstr(),spintf(), sscanf(), atoi() )
- Assignments

**Memory Allocation**

- Overview of memory management
- Allocating new heap memory
- Deallocating heap memory
- Checking for successful memory allocation
- Memory errors
- Using memory you don't own
- Faulty heap management
- Assignments

**Operation in C**

- What are files?
- File naming
- Opening a file
- Writing and reading a file
- Character input and output
- Direct file input and output operations
- Closing and flushing of files
- Sequential and random files
- Assignments

# Data Structures

**Array and pointer:**

- Array creation:
- Array notation(pointer, direct etc)
- Review of pointer fundamentals
- Review of Pointer operations. Parameter passing as pointers.

**Sorting and searching methods**

- Various sorting techniques
  - Bubble sort
  - Merge sort
  - Quick sort
- Various search techniques
  - Sequential search
  - Binary search
  - Radix search

**Stack and queue**

- Stack Fundamentals
  - Stack implementation
- Queue Fundamentals
  - Queue Implementation

**Linked list Fundamentals**

- Link list basics
- Elementary link list functions

**Linked list Advanced**

- Reversing link list(different methods)\
  - By swapping
  - By recursion
- Doubly link list
  - Basics of doubly link list
- Circular link list

**.Trees**

- Binary Tree,
- Complete Binary Tree,
- Recursive and Non-Recursive Traversals in Binary Tree i.e. Preorder, Postorder, Inorder, Binary Search Tree,
- Traversal in BST.

# Operating System Concepts with Linux System Programming

## Theory Session

**Introduction:** Linux Versus Other Unix-Like Kernels, Linux Versions. Basics: Basic Operating System Concepts, An Overview of the Linux kernel and File system. Introduction and Essential Concepts. System 1

Programming: APIs and ABIs Standards, Concepts of Linux Programming, Getting Started with System Programming.

**Process Management:** The Process ID, Running a New Process, Terminating a Process, Waiting for Terminated Child Processes, Users and Groups, Sessions and Process Groups, Daemons.

**Buffered I/O:** User-Buffered I/O, Standard I/O, Opening Files, Opening a Stream via File Descriptor, Closing Streams, Reading from a Stream, Writing to a Stream, Sample Program Using Buffered I/O, Seeking a Stream, Flushing a Stream, Errors and End-of-File, Obtaining the Associated File Descriptor, Controlling the Buffering, Thread Safety, Critiques of Standard I/O. 1

Advanced File I/O: Scatter/Gather I/O, The Event Poll Interface, Mapping Files into Memory, Advice for Normal File I/O, Synchronized, Synchronous, and Asynchronous Operations, I/O Schedulers and I/O Performance. 1

## Lab Session

**Programming problems:** related to processes duplication and replacing, vfork, registering functions, Waiting for Terminated Child Processes, Waiting for a Specific Process, Launching and Waiting for a New Process, Zombies, Preferred User/ Group ID Manipulations, Session System Calls, Process Group System Calls, creating Daemons.

**Programs Using:** Opening Files, Modes, Opening a Stream via File Descriptor, Reading from a Stream, Reading arbitrary strings, Reading Binary Data, Writing to a Stream, Programming Using Buffered I/O, Seeking a Stream, Flushing a Stream, Obtaining the Associated File Descriptor,

Controlling the Buffering, Manual File Locking, Unlocked Stream Operations.

Programming Assignments related to readv( ), and writev( ), Creating aNew Epoll Instance, Controlling Epoll, Waiting for Events with Epoll, mmap( ), Obtaining page size, removing a mapping, Resizing a Mapping, Changing the Protection of a Mapping, Synchronizing a File with a Mapping, Advising, inodes, Sorting by physical block

**Advanced Process Management:** Process Scheduling, Big-Oh Notation, Yielding the Processor, Process Priorities, Processor Affinity, Real-Time Systems, Resource Limits. 1

File and Directory Management: Files and Their Metadata, Directories Links, Copying and Moving Files, Device Nodes, Out-of-Band Communication, Monitoring File Events. 1

Memory Management: Memory Addresses, Segmentation in Hardware, Segmentation in Linux, Paging in Hardware, Paging in Linux, Page Frame Management, Memory Area Management, Non-contiguous Memory Area Management. 2

Memory Management: The Process Address Space, Allocating Dynamic Memory, Managing the Data Segment, Anonymous Memory Mappings, Advanced Memory Allocation, Debugging Memory Allocations, Stack-Based Allocations, Choosing a Memory Allocation Mechanism, Manipulating Memory, Locking Memory, Opportunistic Allocation. 2

**Signals:** Signal Concepts, raising and catching a Signal, Reentrancy, Signal Sets, Blocking Signals, Real-Time Signals, and System Calls Related to Signal Handling. Data structures related to signals. Signal Handlers. Advanced Signal Management, Sending a Signal with a Payload. 1

**Inter-Process Communication:** Pipes, **Fifo** (Named Pipes), Synchronization mechanism- semaphore and MUTEX, Messages Queues, Shared Memory. 6

Programs for Yielding the Processor, Obtaining and Manipulating Process Priorities, Obtaining and Manipulating I/O Priorities, Obtaining and Manipulating Processor Affinity, Setting the Linux scheduling policy, Setting Scheduling Parameters, Determining the range of valid priorities, Obtaining and Manipulating Resource Limits.

Programs for obtaining the metadata of a file,
Obtaining and Manipulating Permissions, ownership, Retrieving, Setting and Listing/ Removing an extended attributes on a file, Obtaining and changing the current working directory, Creating / removing Directories,

Programming problems related to usage and calculation of memory

Writing programs for allocating and deallocating memory space dynamically, allocating arrays, resizing allocations, freeing allocating aligned memory. Programs manipulating pointers, data segments
Creating Anonymous Memory Mappings Mapping /dev/zero, Advanced Memory Allocation using malloc, Fine-Tuning allocated memory, Obtaining Statistics, Duplicating Strings on the Stack, Variable-Length Arrays, Setting Bytes, Comparing Bytes, Moving Bytes, Searching Bytes, Fornicating Bytes, Locking Part of an Address Space, Locking All
of an Address Space, Unlocking Memory.


**POSIX Threads:** Intro to threads, Adv & 2 Drawbacks, Simultaneous Execution, synchronization with semaphores & mutexes, Thread attributes Thread cancellation.

**Input – output control using ioctl:**Introduction to TCP/IP Stack, Introduction to Sockets connections, Network Information, Multiple Clients, IPC with Sockets.

**Interrupts and Exceptions:** The Role of Interrupt and Signals, Interrupts and Exceptions, Nested Execution of Exception and Interrupt Handlers, Initializing the Interrupt Descriptor Table, Exception Handling. Interrupt Handling, Returning from Interrupts and Exceptions.

**System Calls:** POSIX APIs and System Calls, System Call Handler and Service Routines, Wrapper Routines.

**Timing Measurements:** Hardware Clocks, The Timer Interrupt Handler, PIT's Interrupt Service Routine, and The TIMER_BH Bottom Half Functions, System Calls Related to Timing Measurements.

**Process Scheduling:** Scheduling Policy, The Scheduling Algorithm, System Calls Related to Scheduling.

**Kernel Synchronization:** Kernel Control Paths, Synchronization Techniques/primitives,
The SMP Architecture, The Linux/SMP Kernel.

Programming problems related to raising and catching a signal, calls like signal and sigaction, Waiting for a Signal, Mapping Signal
Numbers to Strings, Signal Sets, Blocking Signals, Working with The siginfo_t Structure, si_code, Sending a Signal with a Payload.

Programming problems related to creation of pipes, FIFO, message queues, shared memory. Synchronization techniques like semaphores, pinlocks, and filelocks. Simulating a client server environment using
each of the ipc mechanisms.

Programming problems related to thread creation, attributes, synchronization and other operations, on threads as mentioned above.

Programming problems related to FTPapplication, UDP sockets, TCP sockets, RAW packets, TCP dump.

Programming problems related to generating and interpreting interrupts. Defining an ISR, Installing interrupt handlers, interrupt handling in kernel space/user space.

Programming problems related to importance of system calls like adjtime, alarm, brk, capget, capset, exit, clone, vfork, getpriority, setpriority, ioperm, iopl, kill, modify_ ldtnanosleep, nice, prctl, ptrace, signal stack, sysctl, sysinfo, syslog, uname, access, bdflush

determining the system timer frequency at runtime, tm, Time Source Resolution,

**The Virtual File system:** The Role of the VFS, VFS Data Structures, File system Mounting, Path name Look up Implementation of VFS System Calls, File Locking.

Getting the Current Time of Day, the Process Time, the Current Time of Day, Setting Time with Precision, Adjusting the current time, Sleeping and Waiting, Creating timers, Implementation of PIT, jiffies, timer interrupt handler, dynamic timers, delay functions.

Implementing scheduling algorithms like SCHED_FIFO, SCHED_RR, SCHED_NORMAL, using runqueue data, structurefor scheduling, run queue balancing in multiprocessor systems.

Implementing Spinlocks, seqlocks, RCU, semaphores, completions. Implementation of synchronization strategies for accessing to kernel data structure.

Usage of VFS Data Structures, Implementation of VFS system calls.


# Linux Device Drivers

## Theory Session

**Linux Device Interface:** Device Driver Architecture in Linux, what is a kernel module? Kernel module vs. Application, User space vs. Kernel space, Classes of Devices and Modules.

**Understanding the access method of Device in Linux:** User process Vs Driver, understanding of Linux kernel module, Compiling modules, Loading/unloading modules, Usage C

**Kernel Symbol Table and Parameters:** Understanding and Accessing kernel symbol table, initialization and shutdown, module parameters.

**SCULL:** Understanding the design of scull, implementing scull design.

**Pseudo Driver Development:** Classes of device files: Major and minor numbers, creating device files with mknod, registering character device files, listing character device driver methods.

**Kernel Data Structures:** Detail analysis of character driver and associated data structures and interfaces. Understanding the Linux device model.

**Device operations:** Understanding the access method of Device in Linux: operations on the device like open, release. Understanding the operations on Device in Linux like read, write.

**Memory Allocation and debugging:** Printk for debugging gdb, device information in /proc, kmalloc and DMA memory accessing.

**Concurrency, Race condition and Synchronization:** Concurrency and its management, Semaphores, MUTEXES. Completions, Spinlocks, Semaphores vs Mutex vs spinlocks.Waitqueues.

## Lab Session

Programming problems related to writing hello world module, exporting symbols, modules accepting parameters.

Handling exporting modules to symbol table, interpreting symbols exported by kernel.

Implementation and registration of simple modules and establishing interface between the application and driver. Practicing creation of scull.

Obtaining the driver number and device numbers, interfacing applications to drivers, registering devices, writing a working device driver.

Usage of kernel data structures like file_operations, file, and inode.

Creating the required interfaces and Implementing the device operations like opening and releasing of the device. Creating the required interfaces and Implementing the device operations like read and write on the device.
Usage of debug mechanism like printk, gdb, strace, proc file system.

Implementation of semaphores, MUTEX, spinlocks in device drivers. Usage of completions.

Implementation of ioctl calls like changing Baud rate, changing size of quantum etc.


**Device Control:** Input – output control using ioctl**.**

**Device operations:** Blocking Operation, NoN Blocking Operation, Poll and Select operation.

**Timers and Delays:** Time measurement, Delayed Execution, Determining the current time. Kernel Timers, Tasklets, Work Queues.

**Memory Magement:** Kmalloc, lookaside cache, vmalloc and friends, CPU variables, getting large buffers.

**Interrupt handling and synchronization Techniques:** The /proc interface, Interrupt handlers, Tasklets. Restrictions of kernel code running in interrupt context. Implementing and Installing an Interrupt Handler. Top and Bottom Halves. Interrupt Sharing. Interrupt Driven IO.

**Device Driver development for Serial Controllers:** Layered architecture, Understanding of UART hardware and how the data is transmitted and received using pooling interrupts methods.

**PCI drivers:** PCI interface, look back ISA, sbus, nbus, other PC buses, external buses.

**Memory Mapping and DMA:** mmap device operation, performing direct IO, DMA.

**Block Drivers:** Block Device Architecture, implementation, registration, operation.

Implementing blocking IO, Non-Blocking IO, seeking a device, access control on the device file.

Implementing jiffies, programming using Timer API, Implementation of Kernel Timers, Tasklets, WorkQueues, The Shared Queue.

Practicing using scull, memory pools, get_free_page and friends, implementing the alloc page interface. Implementing scullp: A scull using whole page, scullv: A scull using virtual addresses.

Programming to autodetect the IRQ Number / probing, Implementing a simple Handler, handler with arguments and return value, adding functionality like enabling and disabling interrupts, declare and implement Tasklets, Workqueues. Install and run a Shared Handler, Implement a write buffer mechanism.

Writing A serial driver for for UART hardware like 1655D asynchronous UART with FIFO. Implementing ioctl calls for changing baud rate and FIFO depth.

Registering and Enabling a PCI device.

Implementinga Driver with VMA operations, memory mapping with no page, remapping specific IO regions, remapping RAM, remapping kernel virtual addresses, Implementing data transfer using simple PIC DMA driver.

Writing a complete Block drive including operations like open release, supporting removable media, ioctl method. Implementing request processing, request queues, request completions, block request and DMA.

Implement a complete Network driver with operations such as opening, closing, packet transmission, packet reception, adding a interrupt handler. Implementing a socket buffer, ioctl operation, enable multicast handling.


**Network drivers:** Analysis and understanding of Linux network model, Network device operations, Packet transmission, reception, Interrupt handler, receive interrupt mitigation, changes in link state, Socket buffers, MAC address resolution, and custom ioctl commands, multicast.

- We May Support you for developing some more device drivers, if you are interested and you have time.

**The drivers for which we can support you are as follows:**

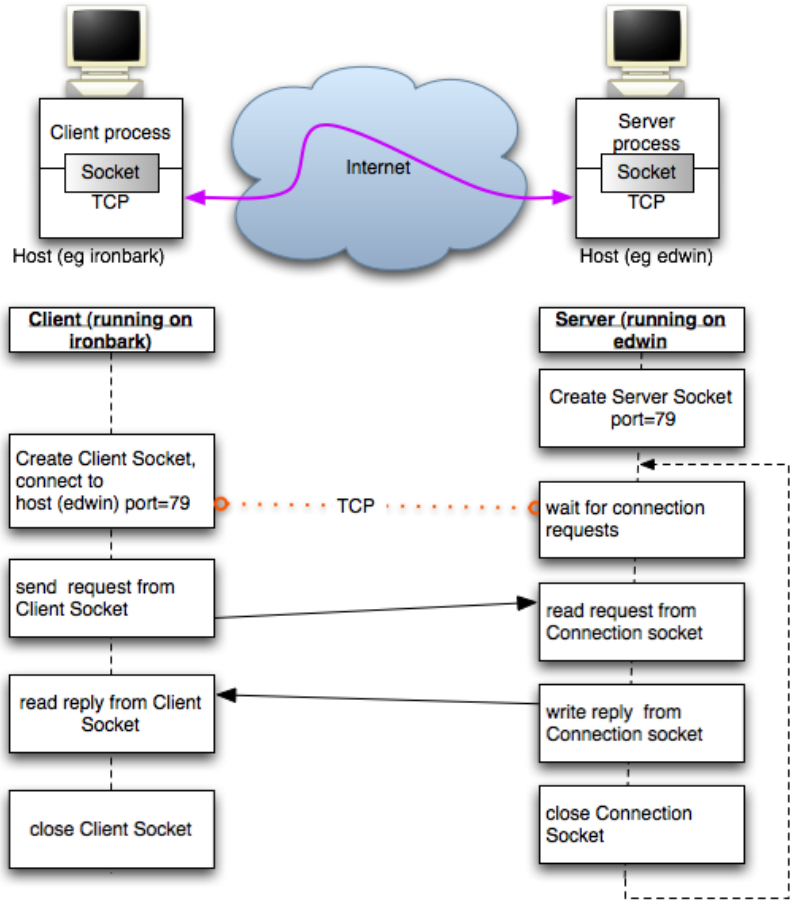
- Input Drivers
- PCMCIA and Compact Flash
- USB Driver.
- Audio Drivers
- Video Drivers
- Infrared drivers
- WiFi Drivers
- RFID Drivers
- Bluetooth Driver
- Cellular Networking
- MAP Drivers
- NAND Chip drivers
- NOR Chip Drivers
- FireWire Driver

- If you have any new devices for which the drivers needs to be developed, we would appreciate the opportunity to contribute to it.

# Projects

**1.Client/Server (TCP/IP) Application Development**

- OSI Model Vs TCP/IP.
- Little Endian , Big Endean
- Developing a program to determine the type of processor of your system.
- Writing a TCP Server Application.
- Sockets?
- Elementary functions for creating a TCP server
  - WSAStartup()
  - socket()
  - bind()
  - listen()
  - accept()
- Writing a TCP Client Application.
- Writing Multithread Chat Server Application. (Optional)
- Writing a TCP/IP application to control a device remotely. (Optional)



**2. Video/Audio Streaming Server using Socket Programming.**

**Introduction**

This article is about a client/server process based socket class. The process is optional since the developer is still responsible to decide if needs it. There are other Server code s
here and other places over the Internet but this one is simple without being effected by any loop hole.

It provides you with the following features-

- No special software required at the client side.
- The software code at the server side is light .
- Streaming is done in local server.
- Any number of clients can connect depends on the server administrator.
- Any format of video can be streamed further only the extension of the video files are to added in the comfit file by the admin.
- Less security issues and robust working