OCS782	EMBEDDED LINUX BSP & PERFORMANCE TUNING	L	T	P	Credits	Total Marks
		2	0	2	3	100

COURSE OBJECTIVES

This course focusses on applying the topics learned in 6th and 7th semester for a real-world application, expands the understanding of the components that make up an embedded system. The course will provide the students with

(i) Advanced C skills and exposure to C++

(ii) The core concepts of boot loader, BSP and performance tuning

(iii) Substantiate the previous learnings of C programming and device drivers,

(iv) the methodologies for debugging and performance tuning

Experiments will run in synchronism with the lectures to support the methods and techniques taught in the lectures. Through the experiments, the students will

(i) learn how to apply driver concepts for real-world use cases

(ii) learn how to setup environment for building and creating custom Linux distributions,

(iii) gain confidence to trouble shoot issues with Linux Kernel

Learning Outcome of students are

(i) Become well versed in C and exposed to C++

(ii) Applying PCI device driver knowledge for a Storage use case

(iii) Understanding of Bootloader and BSP components

(iv) Understanding of build systems

(v) Demonstrate building a custom Linux distribution

(vi) Demonstrate various debugging and performance tuning techniques

TEACHING & LEARNING STRATEGY

Teaching and Learning Strategy	Description of Work	Class Hours	Out-of-Class Hours
Lectures supported by tutorials, assignments, and experiments	Practical in Debian or Ubuntu Linux Environment and an embedded evaluation kit	15 hours	45 hours

PRE-REQUISITES

1. C Programming
2. Linux System Programming
3. Linux Device Driver Programming
4. Embedded Hardware and Sensors (RPI, Beagle bone, EEPROM, RTC, Flash)

UNIT 1 APPLICATIONS of 'C' & C++ BASIC **9Hrs.**

Advanced C: Compiler - linker - loader, ELF, Static and Dynamic Libraries, Structure, Union and Bit field Structure, Pointers, Arrays, Multi-dimensional Arrays, Double pointers, Function Pointers, Memory Allocation, linked list, Queue and Stack, Hash table, Data structure algorithms, Makefiles.

C++ basic: Classes & Objects, C++ Pointers & Arrays, Constructors & Destructors, Overloading, Inheritance, Virtual functions, Polymorphism, Exception handling, Templates, CMAKE.

Lab work

1. Mini project – C based Storage inventory management system
2. Mini project – C++ based client/server

UNIT 2 APPLICATIONS of LINUX SYSTEM PROGRAMMING **9Hrs.**

Industrial Protocols Overview: CAN, Modbus, Profibus, EtherCAT, CoAP

Advanced Process Management: Scheduler Types**,** Priorities, Affinity, Hard Vs Soft Real-time Systems

Advanced Thread Management: Revisiting Thread Models, Patterns, Concurrency, Parallelism, Races, Synchronization and Pthread.

Lab work

1. Implement client/server network program using multi-threading/multi-processing concepts
2. Implement Modbus master/slave using multi-threading/ multi-processing concepts

UNIT 3 APPLICATIONS of DEVICE DRIVERS **9Hrs.**

PCI: PCI Vs PCI-X Vs PCIe, PCI Architecture, Memory and IO map, Configuration Space, Enumeration, Block drivers, DAS/NAS/SAN Overview, Storage Architecture

Ethernet Driver: Ethernet boot loader driver, Ethernet Linux driver, Ethernet tools & tests, Raw socket

General Device Drivers1: Memory, RTC, Sensors, Power rails

General Device Drivers2: Timers, ADC, IOs using PINCTRL, Waveforms using PWM, High level tests for Audio & Video

Lab work

1. Storage application - TBD
2. Demonstrate the Ethernet data and control path using a user space tool.
3. Add an EEPROM to a RPI and demonstrate R/W from a user space program.
4. Add a RTC to RPI and demonstrate alarm from a user space program.
5. Write a user space program to list the power rail spec of the system.

UNIT 4 EMBEDDED LINUX BSP **9Hrs.**

Boot Loader: Introduction, Device tree, Arguments, Boot sequence, Drivers, Optimizations, U-boot commands, Accessing boot environments from user space.

Board Support Package**:** Components of a BSP, Toolchain, build system, Booting Linux locally and from remote, Booting Ubuntu with a new Kernel, Booting with plain vanilla Kernel, Core dumps, Busy box.

Lab work

1. Add a command under U-boot to scan a I2C bus
2. Build RPI image from Yocto
3. Switch Kernel image for every boot

UNIT 5 DEBUGGING & PERFORMANCE TUNING **9Hrs.**

Introduction, Methods and Concepts, Boot parameters, /proc, /Sys and Sysctl, System Monitoring, Timer counter, Memory Management (swap, page, defragmentation), Tuning - memory/Network/Device usage, Controlling Process - Threads, Profiling and Benchmarking methods, Debugging Methods and Techniques - CRO and Logic Analyzers, JTAG, Problem Reproduction, Dumps, Break point, Single Step, variables and history analysis, Print statement, Interrupt, I/O points, Instrumenting code, Code Tracing & Coverage.

Lab work

1. List PCI and USB devices from /sys file system
2. Adjust log level of the system using a virtual file system
3. Adjust display brightness using a virtual file system
4. Control an IO from /proc
5. Demonstrate CPU usage optimization methods
6. Write a program to demonstrate network bandwidth usage
7. Write a program to run two thread and assign them on different processing elements
8. Demonstrate code coverage of user space and Kernel space programs

COURSE OUTCOMES

On completion of the course the student will be able to

CO1. Demonstrate expertise in C++

CO2. Embedded Board bring up

CO3. Create custom Linux distribution for an embedded system

CO4. Analyze issues for an embedded system and improve performance

TEXTBOOK/ REFERENCE BOOKS

Textbooks:

1. Advanced C" by Peter D Hipson & "Advanced C programming by Example" by John Perry
2. C++ Primer by Stanley Lippman 5th Edition
3. Effective modern C++ by Scott Meyers
4. C++ Programming Language by Bjarne Stroustrup
5. Accelerated C++ Practical Programming by Example by Andrew and Barbara
6. Programming: Principles and Practicel by using C++ by Bjarne Stroustrup
7. Linux Device Drivers (LDD), 3rd Edition
8. Linux Kernel Development, 3rd Edition, Robert Love
9. Industrial CommunicationTechnology Handbook by Richard Zurawski

Reference books:

1. Linux System Programming, 2nd Edition, Robert Love
2. The Linux Programming Interface, Michael Kerrisk
3. Linux Kernel Documentation

END SEMESTER EXAMINATION QUESTION PAPER PATTERN

Max. Marks: 100 **Exam Duration: 3Hrs.**

Type of Assessment	Description	Percentage
Quiz (2 or 3)	1. Classroom attention 2. Ad-hoc Quizzes of approx. 40 mins each will be taken to evaluate the continuous progress	15 %
Mid Term (1)	One Mid Term of 1½ hour Practical Examination will be taken	25 %
Laboratory	Laboratory continuous monitoring and Exam	15%
Assignment	Two projects will be given, one before the mid-semester and one after the mid-semester	15%
End Term (1)	4-hour End Practical exam will be taken to evaluate the overall understanding	30 %
	Total	**100%**

OCS783	Linux Kernel & Device Driver Programming	L	T	P	Credits	Total Marks
		2	0	2	3	100

COURSE OBJECTIVES

This course is meant to give them a broad exposure to the understanding of various concepts of Linux Kernel and Device Driver Programming.

The course will provide the students with

(i) The core concepts of operating systems

(ii) the knowledge of Linux Kernel architecture and device driver framework,

(iii) the methodology to write Programs for chosen embedded devices, and

(iv) the skill to use basic techniques for debugging in Linux device drivers and

(v) basic understanding of Linux based embedded system design.

Experiments will run in synchronism with the lectures to support the methods and techniques taught in the lectures. Through the experiments, the students will

(i) learn how to setup environment for developing and debugging device drivers,

(ii) gain confidence to write/customise Linux device drivers, debug and test them using a target platform.

Learning Outcome of students are

(i) Understanding of complex Linux Kernel framework and extending this knowledge to understand any type of operating system

(ii) Writing device drivers for simple embedded devices

(iii) Basic scripting to building the Linux Kernel and Device Drivers

(iv) Demonstrate end-end data flow from device to application

(v) Demonstrate various debugging techniques when the system crashes due to a malfunctioning Kernel or Device Driver

(vi) Plan and execute simple device driver development projects.

Teaching and Learning Strategy

Teaching and Learning Strategy	Description of Work	Class Hours	Out-of-Class Hours
Lectures supported by tutorials, assignments and experiments	Practical in Debian or Ubuntu Linux Environment and an embedded evaluation kit	15 hours	45 hours

Pre-requisites

1. Advanced C Programming
2. Basic Linux utilities
3. Comfortable with Linux text editors (Vim, Gedit, etc.)
4. Experience with major Linux distribution (Ubuntu, Debian, etc.)
5. Free course from Linux Foundation: A Beginner's Guide to Linux Kernel Development (LFD103)
6. Free course from Linux Foundation: Introduction to Linux (LFS101)

UNIT 1 INTRODUCTION TO THE LINUX KERNEL 9Hrs.

Preliminaries - UNIX Vs Linux, Types of Kernels, Versions, Kernel repository, Object-Oriented approach in Linux Kernel, Important Kernel tasks, User-Space and Kernel-Space.

Programming Preview - Error numbers and Getting Kernel output, Task structure, Memory allocations, Kernel data structures, Jiffies

Kernel Architecture Preview - Processes Vs Threads Vs Tasks, Process address space and Process context, Process limits and capabilities, Kernel Preemption, Real Time Preemption Patch, Dynamic Kernel Patching, Porting to a New Platform

Kernel Initialization - System initialization, System boot, U-Boot

Kernel Configuration - Layout of the Kernel source, Kernel configuration files, Kernel Building and Makefiles, initrd and initramfs

Kernel style & general considerations - Coding style, kernel-doc, Using generic Kernel routines and methods, sparse, Using likely() and unlikely(), Writing portable code (CPU, 32/64-bit, Endianness), Writing for SMP, Writing for high memory systems, Power management, Keeping security in mind, Mixing user and Kernel space headers

Modules - What are modules? Compiling modules, modules vs built-in, Module utilities, Automatic loading/unloading of modules, Module usage count, the module struct, Module licensing, exporting symbols, Resolving symbols

Lab work

1. Browsing/searching the Kernel source code
2. Booting a target development board over ethernet
3. Add a new system call
4. Code walkthrough for system initialisation code in Kernel
5. Create your first module and send it for review

UNIT 2 DEVICE DRIVERS I **9Hrs.**

Device Driver Overview - Types of devices, Mechanism vs. Policy, avoiding binary blobs, Power management, How applications use device drivers, Walking through a system call accessing a device, Error numbers, printk(), devres: Managed device resources, Device driver binding

Platform Drivers - Platform devices, I2C subsystem, Device trees, Pin-muxing, Memory management overview, I/O memory and ports

Character Drivers - Character devices, Interaction of user space applications with the kernel, ioctls

Memory addressing - Virtual memory management, Systems with and without MMU and the TLB, Memory addresses, High and low memory, Memory zones, Special device nodes, NUMA, Paging, Page tables, page structure, Kernel Samepage Merging (KSM), Huge page support, libhugetlbfs, Transparent huge pages

Memory allocation - Requesting and releasing pages, Buddy system, Slabs and cache allocations, Memory pools, kmalloc(), vmalloc(), Early allocations and bootmem(), Memory defragmentation

Lab work

1. Implement a module that registers as an I2C driver
2. Modify the Device Tree to list the I2C device
3. Get the driver called when the device is enumerated at boot time
4. Configure the pin-mux for the I2C bus to communicate with the driver
5. Extend the driver to communicate with a I2C device
6. Extend the driver to communicate with user space
7. Extend the driver to implement ioctls
8. Demonstrate driver binding using udev
9. Write a module to gather memory statistics
10. Write a module to create a private memory cache
11. Write a module to setup a memory pool
12. Write a module to check the memory allocation limits

UNIT 3 KERNEL FRAMEWORK I **9Hrs.**

Race conditions & synchronization - Concurrency and synchronization Methods, Atomic operations, Bit operations, Spinlocks, Seqlocks, Disabling preemption, Mutexes, Semaphores, Completion functions, RCU, Reference counts

SMP and Threads - SMP Kernels and modules, Processor affinity, CPUSETS, SMP algorithms (Scheduling, Locking, etc.), Per-CPU variable

Scheduling - Main scheduling tasks, SMP, Scheduling priorities, Scheduling system calls, the schedule() function, O(1) Scheduler, Time slices and priorities, Load balancing, Priority inversion and priority inheritance, The CFS scheduler, Calculating priorities and fair times, Scheduling classes, CFS Scheduler Details

Disk caches and swapping - Caches, Page cache basics, what is swapping, Swap areas, swapping pages In and Out, Controlling swappiness, The swap cache, Reverse mapping, OOM Killer

Signals - What are Signals? Available signals, System calls for signals, Sigaction, Signals and threads, How the Kernel installs signal handlers, How the Kernel sends signals, How the Kernel invokes signal handlers, Real time signals

Lab work

1. Write module to get the current CPU ID
2. Create threads in Kernel and run them in different CPUs
3. Write a Kernel module to launch a user process
4. Write a program to simulate CFS
5. Write a program to defragment memory
6. Write a module to manipulate signals installed from user process

UNIT 4 DEVICE DRIVERS II **9Hrs.**

Input Subsystem - Principles of Kernel Input subsystem, API offered to kernel drivers to expose input devices capabilities to user space applications, User space API offered by the input subsystem

the misc Subsystem - What the misc kernel subsystem is useful for? API of the misc kernel subsystem (both the kernel side and user space side)

GPIO Subsystem - API of the GPIO kernel subsystem (both the kernel side and user space side)

PCI Drivers - What is PCI? Locating PCI Devices, Accessing Configuration Space, Accessing I/O and Memory Spaces, PCI Express

Sleeping and Interrupts, Locking

Monitoring & debugging

Lab work

1. Create a module to handle input events
2. Create a platform driver and hook it to the misc subsystem
3. Write a program to communicate with the GPIO port
4. Create a module to register a simple interrupt handler
5. Make a process sleep and wake-up from a module
6. Handle a variable from 3 simple modules using mutex
7. Handle a variable from 3 simple modules using Semaphore
8. Compute the average time slices of a process
9. Write a program to examine and modify the scheduling policies and priority
10. Write a basic PCI driver
11. Write interrupt example programs from LDD3

UNIT 5 KERNEL FRAMEWORK II **9Hrs.**

DMA - What is DMA? DMA directly to user, DMA and Interrupts, DMA memory constraints, DMA masks, DMA API, DMA pools, Scatter/Gather mappings

Memory Technology Devices - What are MTD Devices? NAND vs. NOR vs. eMMC, Flash filesystems

Block Drivers - What are Block Drivers? Buffering, Registering a block driver, gendisk structure, Request handling

Power Management - ACPI and APM, System power states, Callback functions

Network Drivers - Network Layers and Data Encapsulation, Datalink Layer, Network Device Drivers, Loading/Unloading, Opening and Closing, net_device, sk_buff, Tx/Rx, iocts, NAPI

Overview of ARM support in Kernel

Lab work

1. Write a simple block driver
2. Code walkthrough for an example USB driver
3. Code walkthrough for an example Network driver
4. Write SNULL driver example from LDD3
5. Write DMA driver example from LDD3

COURSE OUTCOMES :

On completion of the course the student will be able to

CO1. Write device drivers for simple devices used in Linux based embedded systems.

CO2. Design and implement simple embedded applications

TEXTBOOK/ REFERENCE BOOKS

Text Books:

Linux Device Drivers (LDD), 3rd Edition

Linux Kernel Development, 3rd Edition, Robert Love

Reference Books:

Linux System Programming, 2nd Edition, Robert Love

The Linux Programming Interface, Michael Kerrisk

Linux Kernel Documentation

END SEMESTER EXAMINATION QUESTION PAPER PATTERN

Max. Marks: 100 **Exam Duration: 3Hrs.**

Type of Assessment	Description	Percentage
Quiz (2 or 3)	1. Classroom attention 2. Adhoc Quizzes of approx. 40 mins each will be taken to evaluate the continuous progress	15 %
Mid Term (1)	One Mid Term of 1½ hour Practical Examination will be taken	25 %
Laboratory	Laboratory continuous monitoring and Exam	15%
Assignment	Two projects will be given, one before the mid-semester and one after the mid-semester	15%
End Term (1)	4-hour End Practical exam will be taken to evaluate the overall understanding	30 %
	Total	**100%**

OCS784	Linux System Programming	L	T	P	Credits	Total Marks
		2	0	2	3	100

COURSE OBJECTIVES

This course is meant to give them a broad exposure to the understanding of various concepts of System Programming.

The course will provide the students with

(i) the concept of advanced Embedded C Language,

(ii) the knowledge of microcontroller architecture,

(iii) the methodology to write Program for chosen microcontroller, and

(iv) the skill to use basic technique for debugging in Linux environment and

(v) basic understanding of system programming.

Experiments will run in synchronism with the lectures to support the methods and techniques taught in the lectures. Through the experiments, the students will

(i) learn how to use Linux commands and programming environment,

(ii) gain confidence to do system programming in Linux environment, and learn how to test and debug the developed programs.

Learning Outcome of students are

(i) Perform advanced C programming using linked list, Function pointers, arrays, sorting, etc.,

(ii) 2. Basic scripting in Linux environment

(iii) 3. Demonstrate programming using Linux System calls

(iv) 4. Demonstrate various debugging techniques when the program throws error

(v) 7. Plan and execute projects simple system programming projects.

Teaching and Learning Strategy

Teaching and Learning Strategy	Description of Work	Class Hours	Out-of-Class Hours
Lectures supported by tutorials, assignments and experiments	Practical in Debian or Ubuntu Linux Environment	15 hours	45 hours

UNIT 1 EMBEDDED C PROGRAMMING 9Hrs.

Essential C Programming Concepts, Variables and Constants, IO operation, Memory Layouts, Operators and Expression, Control Statements, Functions, Arrays, Pointer, Structure and Union, Debugging Methods.

Lab work

1. Programs to understand the various C memory layouts and variable storage locations.
2. Programs to understand the control and conditional statements extensively.
3. Programs to understand the passing of 1-D, 2-D arrays to functions as argument.
4. Programs to understand the pointer usage and passing argument by reference.
5. Programs to understand the structure, union, enum, typedef and bitwise logical operations.

Project

1. Projects based on the modular programming design.
2. Projects based on event handlers
3. Projects based on finite state machine

UNIT 2 EMBEDDED SYSTEM SOFTWARE 9Hrs.

Assembly and C Microcontroller Programming, IO Port Programming, Timer Programming, Handling Interrupts, Interfacing with external peripherals and memory, ADC Programming, LCD & Keypad Programming, Interfacing with Sensors.

Lab work

1. Programs to understand the GPIO input, output and alternate functionalities.
2. Programs to understand the timer functionalities in the microcontroller.
3. Programs to understand various interrupts and interrupt service handler and interrupt service routines.
4. Programs to understand the external interrupts and external peripheral interface with the microcontroller.
5. Programs to understand the ADC and DAC interfaces with the microcontroller.
6. Programs to understand the display and keypad interfaces with the microcontroller.
7. Programs to understand the external sensor signal handling in the microcontroller.

Project

1. Constructing the circuit in the breadboard to execute the lab work exercises in the actual hardware.
2. Developing programs for microcontroller with modular programming using state machines approaches.

UNIT 3 EMBEDDED LINUX SYSTEM PROGRAMMING 9Hrs.

System Programming Concepts, File Operation Programming, IO Operation Programming, Advance Process Management Programming and Thread Programming and Mutex.

Lab work

1. Programs to understand the file management in Linux C.
2. Programs to understand the process handling in Linux C.
3. Programs to understand the threads and mutex in Linux C.

Project

1. Developing multi processes project in Linux C.
2. Developing project using threads and mutex.

UNIT 4 EMBEDDED LINUX SYSTEM PROGRAMMING IPCs 9Hrs.

Essential Inter Process Communication and Synchronization, Pipes, Signals, Timers, Shared Memory, Message Queues and Semaphores.

1. Programs to understand the Pipes in Linux C.
2. Programs to understand the Signals and Timers in Linux C.
3. Programs to understand the Shared Memory and Semaphores in Linux C.
4. Programs to understand the Message Queues in Linux C.

Project

1. Developing event-based project using Linux IPC and synchronization mechanism.
2. Developing message queue-based project using message queue and synchronization mechanism.

UNIT 5 ADVANCE MICROCONTROLLER PROGRAMMING 9Hrs.

Introduction of Advanced Microcontroller Architectures, Buses, Memory, Clock Tree, Reset, Interrupts Management, IOs, Timers, and external interfaces.

Lab work

1. Programs to understand the various clock configurations.
2. Programs to understand the timer configurations and event handlers.

3. Programs to understand various interrupts and interrupt service handler and interrupt service routines.
4. Programs to understand the external interrupts and external peripheral interface with the microcontroller.
5. Programs to understand the handling of external signals.

COURSE OUTCOMES :

On completion of the course the student will be able to

CO1. Write System programs in Linux environment.

CO2. Design and Implement simple system projects

TEXT BOOK/ REFERENCE BOOKS

Text Books:

1. Advanced Embedded C Programming
2. The C Language Programming, Brian W. Kernighan • Dennis M. Ritchie, 1998.
3. Linux System Programming
4. Linux System Programming, Robert Love, 2013.
5. Advanced Microcontroller Programming
6. Discovering the STM32 Microcontroller, Geoffrey Brown, 2012.

Reference Books:

Advanced Embedded C Programming

1. THE FIRMWARE HANDBOOK, Jack Ganssle, 2004..
2. C Programming for Embedded Systems, Kirk Zurell, 2000.
3. Making Embedded Systems, Elecia White, 2011.

Linux System Programming

1. Beginning Linux Programming, Neil Matthew & Richard Stones, 2008.
2. Hands-On System Programming with Linux, Kaiwan N Billimoria, 2018.

Advanced Microcontroller Programming

1. Mastering STM32, Carmine Noviello, 2016.
2. ARM Cortex M4 Cookbook, Dr. Mark Fisher, 2016.
3. RM0090, Reference manual, STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm®-based 32-bit MCUs.
4. PM0214 Programming manual STM32 Cortex-M4 MCUs and MPUs programming manual.

END SEMESTER EXAMINATION QUESTION PAPER PATTERN

Max. Marks: 100 **Exam Duration: 3Hrs.**

Type of Assessment	Description	Percentage
Quiz (2 or 3)	1. Classroom attention 2. Adhoc Quizzes of approx. 40 mins each will be taken to evaluate the continuous progress	15 %
Mid Term (1)	One Mid Term of 1½ hour Practical Examination will be taken	25 %
Laboratory	Laboratory continuous monitoring and Exam	15%
Assignment	Two projects will be given, one before the mid-semester and one after the mid-semester	15%
End Term (1)	4-hour End Practical exam will be taken to evaluate the overall understanding	30 %
	Total	**100%**