|  |  |
| --- | --- |
| **Day** | **Assignments** |
| 01 | **Kernel Compilation**   1. Compile the kernel using Cross Compiler Toolchain. Install the new kernel in the board, boot into the board 2. Write 5 OS assignments on the board |
| 02 | **Introduction to Module Programming**   1. Write the Hello World Module Program. Write the Makefile to compile the Module Program. Compile and load the Module into the Kernel 2. Write a Module program and pass integer type module parameters to it. Print/Display the module parameters on the screen 3. Write a module program and pass 2 integer type parameters and 1 string type module parameters to it. If the string type parameter is add, the module should call a function add() and calculate the sum of the 2 integer type parameters. Similarly, sub, mul and div should be implemented through separate handler functions and perform respective operations |
| 03 | **Module Programming and Exporting Symbols**   1. Write a module program Add.c that exports the add functionality using the EXPORT\_SYMBOL macro. Write another module program Average.c that calls the add function, exported by the Add Module and computes the average of 2 numbers. Manually implement and demonstrate this functionality. 2. Automate the above program so as to be used with modprobe utility to resolve dependencies and load the required modules |
| 04 | **Introduction to Character Drivers**   1. Write a character driver with open and close functionality. Test the driver by writing an application that opens and closes the device driver. When open or closed, the open and close calls in the driver should be executed. 2. Write a character driver with open, read, write and close functionalities. Test the driver through a user application by reading data from the driver and writing data to the driver 3. Write a character driver to dynamically allocate a maj, min no pair from the kernel. Test the same and conclude |
| 05 | **Advanced Character Drivers**   1. Write a character driver that implements the ioctl functionality to demonstrate the calculator functionality. The user application should take input from the user and pass corresponding operations - ‘**+**’,’**-**‘,’**\***’,’**/**’ to the kernel module using ioctl. Based on the operation, ioctl should perform the add, sub, mul and div functions and print results in the kernel 2. Modify the above program to pass a structure to using ioctl to the kernel module. The structure should have 3 elements: integer1, integer2, character. Test and conclude 3. Write a calculator driver in the kernel which performs the following    1. Create 4 Device Numbers – Each device number depicts a specific calculation operation like add, subtract, multiply and divide    2. Implement 8 methods – Read\_Add, Read\_Sub, Read\_Mul, Read\_Div, Write\_Add, Write\_Sub, Write\_Mul and Write\_Div    3. In user space, create 4 device nodes for the 4 device numbers created – /dev/AddDev, /dev/SubDev, /dev/MulDev, /dev/DivDev    4. Write 4 user applications in the user space to test the above. If /dev/AddDev application is run, it should write 2 numbers to the kernel and the kernel should add it and return the sum in the subsequent read. |
| 06 |  |