DreamBig
SEMICONDUCTOR

Disrupting the semiconductor industry.

Summer Training

Course Material

Version 0.5

Contents

1. Introduction:

This course material is intended for students who would like to learn various aspects of embedded software and computer networking.
The course is divided into three sections covering both background concepts as well as hands on exercises:

1. C programing
2. OS concepts (focusing on Linux kernel)
3. Computer Networks

Proposed duration of the course is roughly 6 weeks, assuming that this is taught as a full-day course.

Some understanding of C programing, operating systems and computer networks is required as a pre-requisite.

2. C Programing [1 week]:

2.1 Background Concepts [1.5 days]:

- Basics walkthrough of various programing aspects:
    - https://www.tutorialspoint.com/cprogramming/index.htm
- Various storage classes:
    - https://www.geeksforgeeks.org/storage-classes-in-c/
    - https://www.geeksforgeeks.org/static-variables-in-c
    - https://www.geeksforgeeks.org/what-are-static-functions-in-c/
- C program memory layout:
    - https://www.geeksforgeeks.org/memory-layout-of-c-program/
- Pointers.
- Pointer arithmetic.
- Function pointers.
- Unions and Structures.
- Bitwise Operation:
    - https://www.geeksforgeeks.org/bitwise-operators-in-c-cpp/
- Segmentation fault:
    - https://www.geeksforgeeks.org/core-dump-segmentation-fault-c-cpp

2.2 Hands-on Exercises [3.5 days]:

1. In function "`int displayNumbers(char* buf)`", buf points to a string. Complete the function by adding details below:
    a. Print all integers pointed to by the 'buf'.
    b. Skip character if it is not an integer.
    c. Print missing integers if you find a semicolon (;).
    d. Don't print missing integers if the character after the semicolon is:
        i. Not an integer.
        ii. Less than the integer before the semicolon.
    e. Return number of integers printed.

    For example, the string could be "123;9F80;4mb893;6#2$94"
    Output should be "1 2 3 4 5 6 7 8 9 8 0 1 2 3 4 8 9 3 4 5 6 2 9 4".

2. In function "`int isDupValue(int *ptr, int n)`", ptr points to an integer array between -0 and 100, and n is the number of elements in this array.
    a. Write code which returns 0 if all integer values are unique, or 1 if it finds duplicates.
    b. The code should be optimized for time and space complexity.

3. Write a simple C program to convert big endian to little endian and little endian to big endian.

```
int convertEndian(uint8_t *buf, int n, int type)
```

Where buf contains input data, n is the size bytes of buffer in bytes and type is the current endianness of the data (0=little endian, 1 is big endian). Note that valid values for n are 1, 2, 4, 8 and 16.

4. Find all prime numbers in a given range. Take “min” and “max” numbers as an input and display all prime numbers within this range including min and max.

5. Write a C program to reverse a number. Take the number as an input and display its reverse. For example, if input 5485643, output should be 3465845.

6. Write C code to toggle nth bit of a number. Input both the number and the bit to toggle. Display both the input and output numbers in hex and binary representation.

7. Write C program to check the validity of an IPv4 address. Input IPv4 address as an input string and print whether the address is valid or not. Few invalid IPv4 addresses are 192.168.665.4, 10:3.4.6, 55.43.3, ab.3.4.5.

8. Given pointers to the head of two sorted linked lists, merge them into a single, sorted linked list. It is possible that a head pointer may be null meaning that the corresponding list is empty.

9. Implement a doubly linked list. Given the pointer to the head node of the doubly linked list, reverse the order of the nodes in place. Return a reference to the head node of the reversed list. Print both the original and the reversed linked list. Note: The head node might be NULL to indicate that the list is empty.

10. You are given a pointer to the root of a binary search tree and values to be inserted into the tree. Write a program to insert the values into their appropriate position in the binary search tree and return the root of the updated binary tree.

11. Given five positive integers, find the minimum and maximum values that can be calculated by summing exactly four of the five integers. Then print the respective minimum and maximum values as a single line of two space-separated long integers.

12. Write a C program that takes 5-tuple as user input. 5-tuple includes source IP address, destination IP address, layer 4 protocol type, source port and destination port. Calculate hash over the 5-tuple and search it in a hash table. If the entry is present in the hash table, display it else insert the hash value in the table for next input match. For hashing, use any hash algorithm.

3. OS concepts (focusing on Linux kernel) [3 weeks]:

3.1 Background Concepts [1 week]:

- Overview of operating system and role of the kernel
- Linux kernel basics (https://linuxhint.com/linux-kernel-tutorial-beginners/)
- How kernel programming is different from user mode programing
- Different kernel architectures
- System calls
- Interrupts and interrupt handlers
- Bottom halves and deferring work
    - Softirqs
    - Tasklets
    - Workqueues
- Critical regions and locking
    - Semaphore
    - Mutex
    - Spin locks
    - Atomic operations
    - Read-Copy-Update (RCU)
- Memory Management:
    - Paging and page tables
    - Allocating/freeing memory in kernel (kmalloc, vmalloc, allocating physically contiguous pages, allocating DMAable memory)
    - Slab allocation
    - Mmap and why it is used
- SMP and Threads:
    - SMP Kernels and Modules
    - Processor Affinity
    - Per-CPU variables
- Device driver
    - Basic structure of a Linux device driver
    - Linux loadable modules
- Network drivers:
    - Basic design of a network driver
    - netdevs, skbs
    - Queues, descriptors, doorbells
    - DMA
    - NAPI
    - Simple Tx and Rx path
    - Ethtool and how it is supported by the network driver
- Debugging kernel module

3.2 Hands-on Exercises [2 weeks]:

1. Download a Linux kernel sources, configure, and compile it.

2. Write a simple character device driver kernel module. Print different function names when they are invoked. Also see if it appears in /dev directory:
   (https://olegkutkov.me/2018/03/14/simple-linux-character-device-driver/)
   (https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html)

3. Write a simple kernel module with two .c files. The secondary .c file should contain a print function that prints your name. When the module is loaded, call the print function in the secondary .c from the init_module() from the primary .c file.

4. Repeat exercise #3 but use two separate kernel modules with each have one .c file.
   a. Define a function in the second module and call that function from your first module. If successful, define a function in each of the two modules. Whenever a module is loaded, it will call other module's function if the module is loaded, else it should print "module not loaded".
   b. This is module inter-dependency issue and kernel will not load the module if the other module it depends on is not loaded. How can we load the module irrespective of the other module being loaded or not and simply prints "other module not loaded" message if the other module is not loaded.

5. Declare a NULL pointer in your kernel module. Dereference that NULL pointer and observe the behavior. What system component are invoked when we issue this illegal operation?

6. Write a kernel module that creates two kthreads. Both kthreads should run for one minute before stopping. In each kthread, print thread number after every 10 seconds. We should see total of around 6 prints per thread.

7. Create two threads and bind those threads to two different CPUs. Each thread will increment a shared counter 25 million times before stopping. The shared counter should be initialized to zero. When both kthreads stop, shared counter value should be 50 million. As both kthread will be running simultaneously, so there will be race condition and we will not generally see 50 million counter value in the end. Use following strategies to protect the shared counter:
   a. Use spinlocks
   b. Use mutex locks
   c. Use atomic counter
   d. Run both khreads on the same CPU.

8. Write a deferrable function in your kernel module. Deferrable function are called later when we request to schedule them. We can also schedule them on any cpu. Use following infrastructure to implement a simple deferable function.
   a. Use tasklets
   b. Use workqueues

9. Write a kernel module that allocates dynamic memory from kmalloc and vmalloc. What are the advantages/disadvantages of memory allocated using kmalloc vs vmalloc?

10. Edit the ethernet driver used by your local machine and add few prints. Compile and load the driver and observe these prints.

11. The struct net init_net is an object network namespace structure. A network namespace can contain one or many network devices. Display the names of all network interfaces along with their MAC Addresses and MTU present in init_net object. [Hint: traverse linked list]

12. Write a kernel module that takes network interface name as input argument and displays its associated MAC address and MTU.

13. Write a kernel module that takes function name as input argument and prints the function name whenever that function gets called. Moreover, try to see if we can print function input arguments.

14. Read / Write from a file in kernel module.

15. Write a user space app and a kernel module, send user input data from user-space to kernel module by using IOCTL system call.

16. Do scanf in kernel module to receive user input directly in kernel module.

17. Write user-space app that takes input from user and then pass that string to your kernel module without using memcpy, copy_from_user and friends. Hint: buffer should be shared between user and kernel space.

18. Write two user space app and one kernel module. Create a shared buffer between the two user-space apps. Any data written into shared buffer should be seen by other app. Do not use shared memory libraries. Take advantage of kernel module.

19. Live patch a kernel function. Take a kernel function of your choice e.g. printk or write your own kernel module that have some function definition. Then change the definition of that function at runtime.

4. Computer Networks [2 weeks]:

4.1 Background Concepts [1 week]:

- OSI layer model
    - L2:
        - Ethernet
        - VLAN
        - ARP
    - L3:
        - IPv4 (headers, addresses, subnetting)
        - IPv6 (brief introduction)
        - ICMP
        - NAT
    - L4:
        - TCP/UDP
- Packet Routing:
    - L2 (MAC based)
    - L3 (IP address based)
- Overlay networks
    - VXLAN
    - GRE
    - NVGRE
- IPSec:
    - What is IPsec and why it is used?
    - ESP vs AH
    - Tunnel vs Transport
- Socket programming:
    - Basics of socket programing
    - Type of sockets (TCP, UDP, RAW)
    - Client vs server model

4.2 Hands-on Exercises [1 week]:

1. Use tcpdump or wireshark to look at different headers of the networking traffic (Ethernet, IP, TCP, UDP, ICMP).

2. Write a simple C program to create an Ethernet header with Ethernet MAC, IP header and UDP using the data below – dump the packet to verify various header contents:

    src mac : 04:4a:df:aa:00:11
    dst mac : 04:4a:00:bb:ab:44
    src IP : 192.169.2.5
    dst IP : 192.169.2.20
    dport : 124
    sport : 555

payload : "This is a test packet generated by me"

Hint: Understand MAC, IP and UDP header fields. Use default data for protocol type, version and other header fields. Calculate the header length field correctly.

3. Write C code to parse the Ethernet header and print each field separately. Parse Ethernet MAC, IP and UDP header. (You can pass the packet as an input parameter to the function).

4. Write C code to detect whether the packet received is broadcast, multicast or unicast by looking at its MAC address (You can pass the packet as an input parameter to the function).

5. Write a client-server socket program in C language using localhost and TCP socket. Anything sent by the client should be received and displayed by the server. Server should send a unique acknowledgment number back to the client after each received message. Client should display this number.

6. Setup an IPsec tunnel between two machines. You can use either IKE or manual keying for the tunnel setup. Run ping between the two machines to verify that both machines can talk to each other.
    a. Bonus: Setup a third machine in between. This machine will have two interfaces, one connected to each IPsec machine. This third machine will forward packets between its two interfaces. You can dump packets on one of its interfaces to see encrypted IPsec packets.

7. Write a RAW socket application which can respond to a ping request from another machine:
    a. https://www.cs.dartmouth.edu/~sergey/cs60/lab3/lab3.pdf