New Jersey Institute of Technology
Digital Commons @ NJIT

Computer Science Syllabi | NJIT Syllabi

Fall 2020

CS 433-001: Introduction to Linux Kernel Programming

Andrew Sohn

Follow this and additional works at: https://digitalcommons.njit.edu/cs-syllabi

Recommended Citation
Sohn, Andrew, "CS 433-001: Introduction to Linux Kernel Programming" (2020). *Computer Science Syllabi*. 138.
https://digitalcommons.njit.edu/cs-syllabi/138

This Syllabus is brought to you for free and open access by the NJIT Syllabi at Digital Commons @ NJIT. It has been accepted for inclusion in Computer Science Syllabi by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.

CS433 Intro Linux Kernel Programming - Course Syllabus - Fall 2020

- Class Web page: http://web.njit.edu/~sohna/cs433 and http://canvas.njit.edu
- Instructor: Andrew Sohn, GITC 4209, (973)596-2315, email: sohna at njit dot edu
- Office Hours:
  - 3:30-4:30 pm, Mon, on Webex,
  - 1-2 pm, Thur, in person, either in my office or larger room to be announced. Until further notice, come to my office for in-person office hours. Make sure to have a face-covering mask on while maintaining 6 feet distance.
  - by appointment. If you want to see me outside the office hours, send me an email.
- **Teaching assistant:** not assigned at the moment (Mon, 9/1/2020) and it's not certain we'll get one as very few qualify for this course.
- **Grader:** trying to get one but again very few qualify for this course.
- **Class time and location**: Mon, 12:30-3:20 pm, Webex.
- **Kernel version 5.9-rc3 as of 9/1/2020:** download the latest version at kernel.org
- **Books recommended**:
  - W. Mauerer, Professional Linux Kernel Architecture, 2008, Wiley, ISBN:9780470343432 (10 years old but it's still "the latest" or "the most recent"). I originally required this book but the bookstore told me they are unable to procure as it is out of print, and hence I was suggested to change required to recommended. So it's officially recommended. Go get a copy if you can find one.
  - Understanding the Linux Kernel, Third Edition, 2006, Bovet and Cesati, O'Reilly, ISBN: 0-596-00565-2 (very old but you still learn a lot).
  - Intel 64 and IA-32 Architectures Software Developer's Manual, only 5038 pages :-) as of 9/6/2018. https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf
  - CS350 textbook: Computer Systems: A Programmer's Perspective, 3/E (CS:APP3e), Randal E. Bryant and David R. O'Hallaron, Pearson (July 6, 2015), ISBN-13: 978-0134123837, ISBN-10: 0134123832.
- **Grading**:
  - Attendance (4%)
  - Programming assignments (20%) - submit on Canvas
  - Test1 (20%), 12:30-2:00 pm, Mon, 10/5/2020 on Canvas - **See below for taking exam on Canvas.**
  - Test2 (20%), 12:30-2:00 pm, Mon, 11/9/2020 on Canvas - **See below for taking exam on Canvas.**
  - Final exam (36%) on Canvas, Date and Time TBD, See the registrar's page. - **See below for taking exam on Canvas.**
- **Attendance:** I am required to verify your presence. The first thing you do when you get on Webex is type in "last name, first name, here" in the chat pane. I will scrape this chat record as attendance. The name you type in must match your Webex ID. If not matched, you'll be marked absent for that lecture. Character case will be ignored.
- **Homework**:
  - Homework is posted on http://web.njit.edu/~sohna/cs433
  - See Canvas for HW due dates and submission.
  - Homework is due at 11:59 pm of the posted due date.
  - Homework will not be accepted after the due date. Submit on time. Do not ask for exceptions. If you ask for an exception, I will apply that to everyone in class to be fair with every one else in class.
  - Do your homework from scratch and on your own. Be prepared to spend two hours a day on homework.
  - Homework must be your own work. Do not show your code and/or copy other's code.
  - Copying homework will be referred to the University for disciplinary actions.
  - If you are unable to do the first three homework (which are simple and mechanical ones), this class is not for you.
- **Setting up lxr:** linux cross referencer for source code traversing. See http://lxr.sourceforge.net/en/index.php for setup instructions. If you are unable to set up lxr on your box in the first two weeks, this course is not for you as you need lxr to study the source code (tens of millions of lines of C and assembly) and do homework.
- **Taking exam on Canvas. Read carefully:**
  - Sample exams will be posted on the class web site for your reference. Do not rely soley on them. The contents and format can/will be completely different.
  - Exam questions will be given out in a random order with multiple versions of the same difficulty/complexity. Once you submit, you can NOT go back to fix or edit. Exam will start and end at the designated date and time

sharp. You will receive a message regarding the number of exam questions and their points before exam. Budget your time accordingly.
    - **Disagreement with exam marking/scores**: If you disagree with your exam scores/marks, you may dispute within a week of receiving/seeing the graded exam paper. After a week, no exams will be contested.
    - **Grading dispute**: If you disagree with your grade, you may contest after the first day but within a week of the following semester. After a week of the first day of the following semester, no grading dispute will be considered.
    - **NJIT policy on missed exams**: There will be no make-up exam(s). You must plan your semester accordingly, especially if you work. Should you miss the exam(s) due to emergency, (a) go to/contact the Dean of students, (b) explain your situation as to why you had to miss, and (c) ask to issue a memo to me. If and when I receive a memo from the Dean on your missed exam, I will copy your next exam score to the missing one. Those who miss the final exam will fail in the course unless you demonstrate a true emergency again through the office of the Dean of students. No other policy will be applied. No exceptions will be made.
- **Academic Integrity:** I am required to post this on the course syllabus.
"Academic Integrity is the cornerstone of higher education and is central to the ideals of this course and the university. Cheating is strictly prohibited and devalues the degree that you are working on. As a member of the NJIT community, it is your responsibility to protect your educational investment by knowing and following the academic code of integrity policy that is found at: http://www5.njit.edu/policies/sites/policies/files/academic-integrity-code.pdf. Please note that it is my professional obligation and responsibility to report any academic misconduct to the Dean of Students Office. Any student found in violation of the code by cheating, plagiarizing or using any online software inappropriately will result in disciplinary action. This may include a failing grade of F, and/or suspension or dismissal from the university. If you have any questions about the code of Academic Integrity, please contact the Dean of Students Office at dos@njit.edu"
- Invitation to your social network(s): Do not ask to be connected to your social network. My policy is not to join anyone or any social networks for any reason. Once it's out, it will be out there forever. You will never be able to take it back. Do it at your own risk.

Lecture Schedule by Week (may change based on class pace)

1. Introduction: Lecture Note 1
    - LAMP, virtualization, containers, datacenter computing infrastructure,
    - Review of Intel architecture based on Intel 64 and IA-32 Architectures Software Developer's Manual (4684 pages! as of 1/16/2018),
    - Short recap of Intel and AT&T Linux assembly
    - "Hello, World!" in Intel assembly, Linux assembly, assembly in-line programming with C
    - Setting up LXR (Linux cross referencer http://lxr.sourceforge.net/en/index.php) on your laptop
2. Preparatory steps: Lecture Note 2
    - Compiling the kernel, which will take hours the first time
    - Module programming - see The Linux Kernel Module Programming Guide
    - Writing your own system calls, adding to the syscall_64.tbl
    - Booting - machine BIOS, disk MBR, Grub Linux loader, preliminary setup (setup(), startup_32/64() 1 and 2)
    - Overview of kernel startup and initialization - start_kernel()
3. Memory - initialization
    - Overview of memory spaces: logical segmentation, linear virtual, actual physical
    - Detecting BIOS-provided e820 physical RAM map: detect_memory()
    - Converting to memblocks: setup_arch(): e820__memory_setup(), e820__end_of_ram_pfn(), e820__memblock_setup(), init_mem_mapping(), initmem_init()
4. Memory - paging, buddy system, setting up page directories (global, upper, middle), tables and PTEs
    - (N)UMA, nodes, zones (DMA .. Normal), memory types (Unmovable .. CMA .. Isolate), free_areas
    - Setting up buddy system: x86_init.paging.pagetable_init(), paging_init(), zone_sizes_init();
    - Allocating 1 to 1K contiguous pages from buddy system: __get_free_pages() to __rmqueue_smallest()
    - Freeing pages: free_pages() to __free_one_page()
    - Setting up slabs for small memory objects of 8 bytes to 8 KB: mm_init() to kmem_cache_init() for initializing general purpose kmalloc_caches():

5. **Test 1, 12:30-2:00 pm, Mon, 10/5/2020 on Canvas**

   Memory - setting up kmalloc_caches (slabs) for small objects of 8 bytes to 8 KB to large non-contiguous memory space
   - Setting up slabs for small memory objects: mm_init() to kmem_cache_init() for initializing general purpose kmalloc_caches():
   - Allocating small memory chunks: kmalloc() of 8 bytes to 8 KB from slabs
   - Freeing small memory chunks: kfree() to return to slabs
   - Large non-contiguous memory, process address space: vmalloc() and vfree() if time permits
   - User malloc(): will discuss briefly or provide pointers to read on your own as it's simple compared to __get_free_pages and kmalloc if time permits.
6. Process - structrures, organization, initialization
   - Structures: thread union, thread info, stack, task, and thread struct, PID0 (swapper)
   - Macros to initialize PID0: INIT_THREAD_INFO(init_task), INIT_TASK(), INIT_TASK_TI(), ...
   - Initial hardcoded structs: init_task, init_stack, init_mm, init_fs, etc.

   Process - creating the first five kernel threads
   - Creating kernel threads: kernel_thread() - copy and insert insert into RB tree, create_kthread(), hot plug thread
   - P0 creating PID1 (init) and PID2 (kthreadd) using kernel_thread(),
   - P1 and P2 creating PID3 (softirqd), PID4 (migrationd) using create_kthread() through kthread_create_list
   - Hot plug threads for creating P3 and on using kthread
   - Sync mechanisms between P0, P1, and P2 for creating P1 and P2; and between (P1,P2) and (P3,P4,...) for creating P3 and on.
7. Process - process scheduling (do_fork() to schedule() to rb_entry())
   - update_curr(): priority, nice value, weight, delta, weighted delta, actual runtime, virtual runtime
   - schedule() - configurable scheduling policies, fair_sched_class, computing vruntime based on priority (nice values) and weighted delta.
   - Scheduling processes with red-black tree: pick_next_task(), put_prev_task;

   Process - process switching (schedule() to __switch_to())
   - Context switches
   - Switching to suspended process
8. Process scheduling and switching continue

   Interrupts - PICs, APICs, exceptions (traps) and hard interrupts, IDTs
   - Hardware organization: Programmable Inerrupt Controller, interrupt vectors, CPU interrupts and interrupt acknowlege to interrupt handlers.
   - Initialization: sort_main_extable(), trap_init(), init_IRQ(), softirq_init(), initializing IDTs, irq_desc and softriq_vec
   - Exceptions: exception handlers, do_trap() to do_exit() if no trap handler
   - Hard interrupts: registering interrupt handlers request_irq() to do_IRQ() to handle_level_irq(), edge vs level trigger
9. Interrupts - timer interrupts, soft interrupts, ksoftirqd, run_timer_softriq
   - Hard interrupts: IDT handler common_interrupt, interrupt_entry, do_IRQ(), device driver action->handler(), raising softirq invoke_softirq(), softirq_vec action
   - Soft interrupts: hardirq (schedule,producer,top half) vs softirq (action,consumer,bottom half)

10. **Test 2, 12:30-2:00 pm, Mon, 11/9/2020 on Canvas**

    Interrupts contiune - PICs, APICs, exceptions (traps) and hard interrupts, IDTs
    - ksoftirqd (kernel thread 3, P3), do_softirq(), softirq action, run_timer_softirq() as an example

    NOTE: At this moment, I find that most of you are completely overwhelmed. It is expected because memory, processes and interrupts are the three pillars of the kernel. Oftentimes, I find only two weeks left, instead of four as originally scheduled here since I have to repeat due to its sheer code volume, difficulty and complexity all combined. But the two topics below, file systems and networking, are exciting because they are the two applications of the kernel, the applications of what you have learned so far. However, they are complex and

difficult. More often than not I find myself shortening the two topics. Or in some cases, only file system is discussed. In in some years, I cover two topics lightly because I have two weeks left instead of four. So be warned. We'll see how it goes this semester.

11. File system - virtual file system, block IO, elevator scheduler, device driver, softirq, timer, delayed work, kblockd_workqueue
    - initialization: vfs_caches_init, mnt_init: Virtual file system VFS
    - Registering, mounting
    - Scheduler - completely fair queuing
12. File system - device driver, Ext4 example (vfs_read() to to scsi_dispatch_cmd())
    - submit_bio(), scheduler
    - The big loop in time and space: timer, delayed work, kblockd_workqueue
    - Ext4 disk organization: MBR, superblock, group descriptors, bitmaps, inode table, data blocks
13. Networking - receiving packets
    - receiving packets: softnet data, input_pkt_queue, process_queue, budget
    - receiving packets: NIC, ISR, Softirq, IP, TCP, Inet, BSD, User
    - User BSD sockets read(), tcp_recvmsg, , Inet socks, TCP and IP layer
    - Softirq: net_rx_action, ip_rcvmsg, tcp_recv_msg
14. Networking - sending packets
    - sending packets: output_queue, User, BSD, Inet, TCP, IP, Softirq, Qdisc, ISR, NIC
    - User BSD sockets, Inet socks, TCP and IP layer
    - Qdisc - p/bfifo packet/byte based FIFO queueing discipline, ISR interrupt service routine, NIC
    - Softirq - dequeue, transmit, ISR interrupt service routine, NIC, requeue, delayed timer
15. **Final exam (week15):** See the registrar's page: http://www.njit.edu/registrar

Read the following carefully to make an informed decision on whether this course is for you:

- This class is extremely difficult and time consuming. You are at your own risk.
- No lecture notes are available for this course other than the two posted at the class web page. I used to prepare lecture notes in electronic format for distribution, spending months and months, only to find that they become obsolete in a semester or two. I don't do that anymore, knowing full well that it's likely futile in the near future.
- There is little literature available on the Web for the current kernel that consists of tens of millions of lines of C and assembly.
- No textbook is available that discusses the current contents, say v5.8 as of 9/1//2020. No authors in their right mind would spend years to write about the topics only to find that the contents have changed in the mean time and half the textbook might become obsolete. This is assuming that the authors are capable of fully understanding million lines of C and assembly code.
- Usually about a half the class stays till the end due to its rigor and sheer amount of work involved.
- It is an elective course and if you decide to stay despite my warnings you are held accountable for the course materials.
- Granted some OS fundamentals stay the same over time but this course is not about the underlying theories but rather about a real operating system that powers the Internet, billions of cell phones, the Internet of Things, car dashboards, airplane entertainment systems, etc.
- It has been suggested time and time again that this course should be required for every computer science student because everything you learned in computer science and a lot more are working together to make it happen.
- If you follow what's discussed in class and do the homework _on your own from scratch_, you would empower yourself with advanced knowledge few have which could lead to long term security in computing as the entry barrier to the Linux kernel is high, to say the least.

Representative comments by the students in the past 14 years:

- This is the most difficult course in my life.
- This is the most exciting course in my life.
- You need a lot more than what you learned in your life.
- Somebody can change the world.
- This is not for the faint of heart.